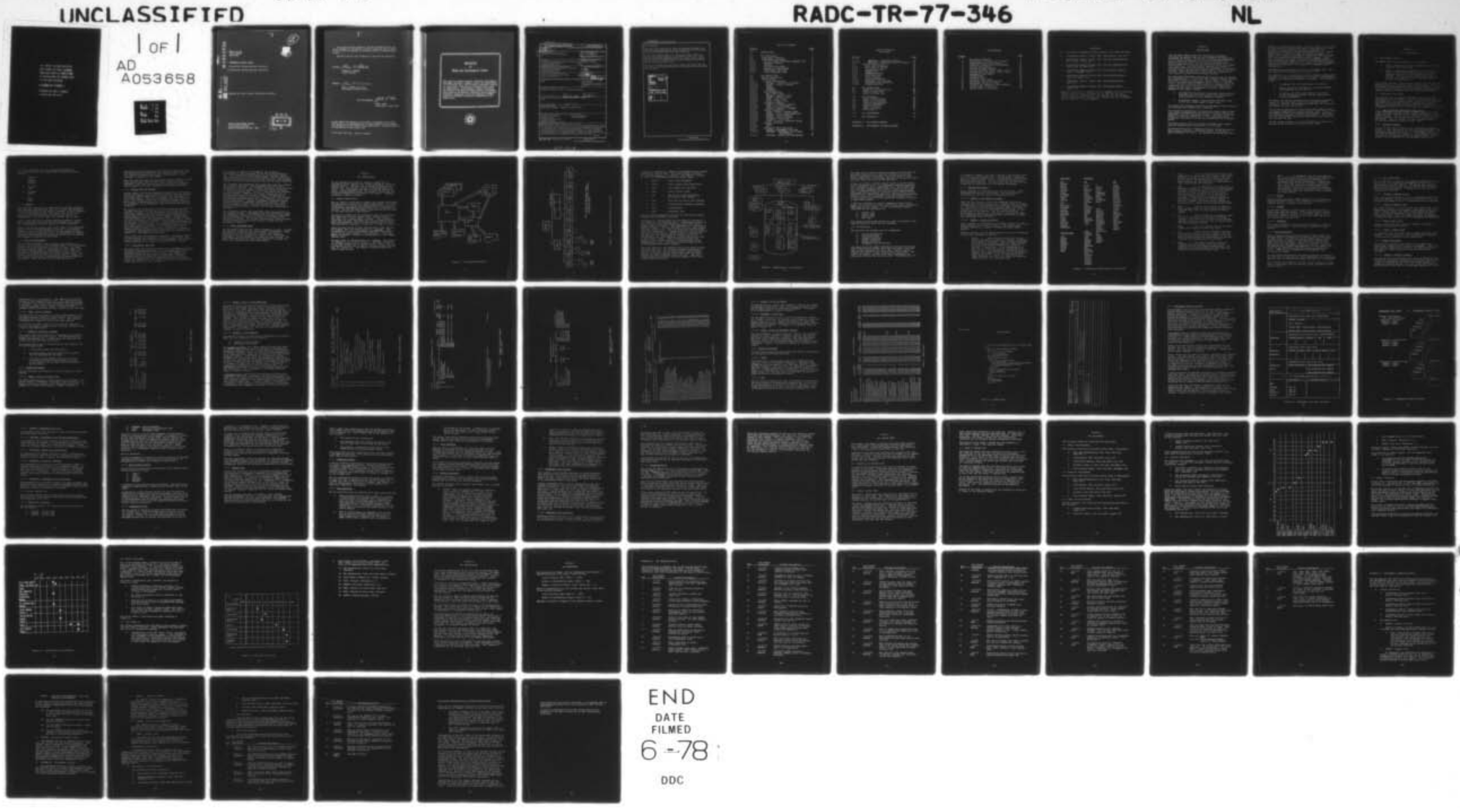
AD-A053 658   IBM FEDERAL SYSTEMS DIV GAITHERSBURG MD   F/G 9/2
PROGRAMMING SUPPORT LIBRARY.(U)
MAR 78   F30602-76-C-0016
UNCLASSIFIED   RADC-TR-77-346   NL

1 OF 1
AD
A053658



END
DATE
FILMED
6-78
DDC

RADC-TR-77-346
Interim Report
March 1978

PROGRAMMING SUPPORT LIBRARY

International Business Machines Corporation

International Business Machines Corporation

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-77-346 has been reviewed and is approved for publication.

APPROVED: Lawrence M. Lombardo

LAWRENCE M. LOMBARDO
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM, Asst Chief
Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19 **REPORT DOCUMENTATION PAGE**		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 RADC-TR-77-346	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* 6 PROGRAMMING SUPPORT LIBRARY.		5. TYPE OF REPORT & PERIOD COVERED 9 Interim Rep[redacted]t.
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) International Business Machines Corporation Federal Systems Division Gaithersburg MD 20760		8. CONTRACT OR GRANT NUMBER(s) 15 F30602-76-C-0016 NEW
9. PERFORMING ORGANIZATION NAME AND ADDRESS International Business Machines Corporation Federal Systems Division Gaithersburg MD 20760		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 55500823
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIM) Griffiss AFB NY 13441		21 12. REPORT DATE Mar[redacted]78
		13. NUMBER OF PAGES 74 12 83 p.
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Same		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

PE63728F 16 5550 17 08

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

WU RADC 55500823

18. SUPPLEMENTARY NOTES

RADC Project Engineer; Lawrence M. Lombardo (ISIM)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Structured Programming
Top-Down Design
Software Development Tool
Computer Software
Software Configuration Control
Top-Down Integration
Top-Down Testing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Programming Support Library (PSL) is a software package designed to provide the facilities for organizing and controlling the development and maintenance of computer programs. The structure and operation of the PSL is oriented toward supporting structured programming techniques (i.e., Top-Down Design and Integration, Use of Structured Source). The PSL has COBOL, FORTRAN, and JOVIAL precompilers to accommodate structured source code input.

(Cont'd)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

174-950

Item 20 (Cont'd)

This report briefly describes the initial PSL development philosophy, the basic functional capabilities and current configuration of the PSL, and the supporting documentation which can be made available.

The PSL has been made available to the World-Wide Military Command and Control System (WWMCCS) community for installation at six sites. It also has been installed at selected non-WWMCCS sites, namely: HQ AFSC (Andrews AFB, MD), Rome Air Development Center (Griffiss AFB NY) and Air Force Data Systems Design Center (Gunter AFS AL).

The PSL is operational on the Honeywell HIS 6180 computer system under the GCOS operating system and the Honeywell HIS 6080 computer system under WWMCCS operating system.

Table of Contents

1

Table of Contents
(Continued)

ILLUSTRATIONS

REFERENCES

*A. Structured Programming Series, Volumes I-XV, RADC-TR-74-300

B. Programming Support Library (PSL) Users Manual, June 1977

C. Programming Support Library (PSL) Test and Implementation Plan, Phase I, June 1976

D. Programming Support Library (PSL) Test and Implementation Plan, Phase II, March 1977

E. Programming Support Library (PSL) Test Analysis Report, Phase I, November 1976

F. Programming Support Library (PSL) Test Analysis Report, Phase II, August 1977

G. Programming Support Library (PSL) Installation Manual, June 1977

H. Programming Support Library (PSL) Maintenance Manual, August 1977.

*RADC-TR-74-300, Vol I, A016771, Vol II, A018046, Vol III, A013255, Vol IV, A015794, Vol V, A003339, Vol VI, A007796, Vol VII, A008639, Vol VII, A016414, Vol VIII, A016415, Vol IX, A008640, Vol X, A008861, Vol XI, A016416, XII, A026947, Vol XIII, A020858, Vol XIV, A015795, Vol XV, A016668.

Section 1

INTRODUCTION

Over the past several years the structured programming technology has gained wide use and acceptance throughout the Department of Defense. The concept of structured programming is 1) to define a program with a single entry point and a single exit point, and 2) to code in such a manner as to develop lower level functions as pieces which plug into higher level functions. By significantly reducing the number of interfaces, potential error points are eliminated. Structured programs tend to be less error prone, easier to read, easier to maintain; these characteristics obviously are attractive to cost-conscious projects with software contents.

The structured programming discipline requires tools to support the implementation. In 1974-1975 IBM's Federal Systems Division delivered to Rome Air Development Center (RADC) a 15-volume series, The Structured Programming Series, which 1) describes a set of structure figures which enable the implementation of structured code, and 2) defines tools and techniques which support the structured programming discipline. Those tools germane to this report include:

a. precompilers which accept structured source code and translate the structure figures into unstructured code acceptable to the respective compiler

b. programming support libraries which maintain code at the unit (single page of code) level

The series also includes guidelines on management data collection and reporting and documentation standards.

The structured programming discipline has further implications than coding a program using the defined structure figures and segmenting code into hierarchical units. Top-down, structured design is an obvious forerunner of structured programming. Integration and testing can also be accomplished in a top-down manner.

Structured source code can be made so readable that certain detailed documentation requirements can be waived.

The World-Wide Military Command-and-Control System has and will add to their inventory a substantial amount of software at bases throughout the world. The problems of maintaining that software are already significant. With the intent of providing the best possible software with ease of maintenance as a major criterion, an experiment was defined to implement and subsequently evaluate structured programming at five selected WWMCCS sites on different types of software projects. To support the structured programming experiment, WWMCCS charged RADC with procuring a programming support library. RADC, in turn, contracted with IBM to develop a programming support library which would operate in the WWMCCS environment and provide the necessary capabilities to support the new programming technology as defined in the Structured Programming Series (Reference A).

The WWMCCS PSL development was begun in September 1975. The full system test was completed on June 14, 1977.

The WWMCCS PSL is a comprehensive system software package which supports the growth and maintenance of structured programming projects in a top-down development environment. The system provides:

a. A framework for the organization of a project

b. Simple functional statements to interface between the programmer and the machine

c. Structural and statistical reports for control of the development process and for communication between programmers.

The WWMCCS PSL system provides special structured programming support for the Structured COBOL and Structured FORTRAN languages. However, unstructured programs may also be stored and maintained under the system.

The PSL system operates on an HIS Series 6000 computer under the General Comprehensive Operating System (GCOS) using a standard WWMCCS software and hardware configuration. Libraries are maintained on direct access storage devices. The system utilizes the standard card reader and printer, one tape drive, and 37K words of core.

The PSL system operates in the batch mode and is directed to perform specific functions by PSL Function cards.

Section 2

PSL DEVELOPMENT

2.1 Development Schedule

The WWMCCS PSL has been developed in two phases:

a. Phase I provided the basic library capabilities and supported structured COBOL

b. Phase II added some capabilities to the library support, provided management data collection and reporting capability, and supported structured FORTRAN. (The structured FORTRAN support was not part of the original contract, but was added as a Phase II requirement).

The original contract period covered eighteen months with nine months allocated to the development of each phase. Technical development was performed from IBM offices in Rosslyn, Virginia, via two Texas Instrument 700 series terminals from which the support computer at Andrews Air Force Base could be dialed.

2.2 Development Philosophy

Not only does the PSL support structured code and top-down integration, the PSL itself is written in structured COBOL and was integrated in a top-down manner. Since no facilities were available on the Honeywell 6000 to support this approach (which is the reason for developing the PSL), the early PSL development was not able to follow the full top-down discipline.

2.2.1 Use of Program Production Library (PPL)

A basic Honeywell library system, Program Production Library (PPL), had been developed by IBM during performance of the FORSTAT project. The PPL consists mainly of existing Honeywell utility programs which enables source code to be stored and maintained. Jobs are spawned to compile program modules. The PPL was utilized to develop Phase I PSL.

2.2.2 Structure Figures

As part of the effort supporting the Structured Programming Series, a COBOL precompiler was specified and developed. The precompiler was made operational on the Honeywell 6000 so that structure programming was possible. This COBOL precompiler was used as a standalone program to compile Phase I program modules and was subsequently incorporated into the PSL.

The COBOL precompiler and all subsequent precompilers incorporated into the PSL support the following structure figures:

a. CASENTRY
CASE
ELSECASE
ENDCASE

b. DO WHILE
ENDDO

c. DO UNTIL
ENDDO

d. IF
ELSE
ENDIF

2.2.3 INCLUDE

True top-down integration is enabled by the INCLUDE statement. This directive indicates that the unit of code (approximately 50 lines) identified by the INCLUDE statement is to be compiled as if it were coded in-line. The source code of the unit containing the INCLUDE statement need not be altered to incorporate, at any later time, the INCLUDEd unit of code.

Since PPL was not able to handle INCLUDE statements, Phase I PSL is coded using the COBOL PERFORM statement which causes the indicated unit to be executed as an in-line function.

Phase I PSL was the library under which Phase II was developed. Phase II code is distinguishable by the use of the INCLUDE statement. The decision was made to retain unchanged Phase I code as originally structured since the replacement of PERFORM statements with INCLUDE statements was costly and the benefits were cosmetic as opposed to functional.

2.2.4 Top-Down Development

The top-down development and integration was adhered to, however, in the design of each program module, in the subsequent coding, and in the integration of units of code. The same discipline was in effect in integrating the PSL system. The top-level program module is Batch-Control, BCTL. BCTL was designed in a top-down manner and coded, top-unit first. As soon as the top unit of BCTL was entered into the library, it was compiled and executed, thus beginning system integration immediately. When the next unit of code was ready, a compilation of BCTL took place including the lower-level unit. The testing which followed not only tested the new unit, but also served as regression testing for the top-unit. BCTL continued to develop in that manner.

When BCTL operation was such that other modules could be called, those modules were added to the system as the units were completed. Each test on each new unit served as a regression test on higher-level units throughout the system hierarchy.

2.2.5 Design and Code Reviews

Another aspect of the new programming technology is the formal review, both design reviews and code reviews. The PSL development team conducted both types of reviews, but frequently on an informal basis. Since the development team was relatively small, the informal review was satisfactory in many cases. A more formal review procedure was implemented during the Phase II development of the Management Data Collection and Reporting Subsystem.

The original PSL architecture and system design was the work of a single designer with informal review and critique by the department manager. As the development team increased in size and as many capabilities were being designed simultaneously, design sessions were held for early exchange of ideas and to take advantage of the "brainstorming" effect. Module design was then solidified into Program Design Language (PDL). Design reviews were conducted on the PDL. Copies were distributed to the reviewers two days before the scheduled review so that each participant would be prepared. The designer whose work was being reviewed conducted the review. For review of less complex functions, two reviewers participated. For review of for example, the Management Data Collection and Reporting Subsystem, all team members participated.

Code reviews were less formally conducted. In general, copies of the code were distributed to two other group members and written comments were returned to the programmer. The technical leader participated in almost all design and code reviews.

2.2.6 Programming Librarian

The PSL project utilized the services of a programming librarian throughout program development. The librarian began using the previously developed PPL. During Phase I development the external library was maintained as closely as possible to a library which fully supported structured code. Although separate listings could not originally be obtained for code units, the librarian cut the longer listings at the appropriate places and filed each unit separately.

At the start of Phase II development, the librarian participated in the conversion to the use of PSL instead of PPL. This required the development of new external library procedures as well as internal procedures. PSL Functions were used to provide library maintenance for Phase II. The internal library was saved at least weekly using the PSL BACKUP Function.

The librarian was responsible for maintaining both the internal and external libraries. Each programmer had one or more development libraries. The librarian maintained these libraries as directed by the programmer, adding new code and changing or deleting existing code. Daily an Index Report was obtained for each library. Whenever an external library was out of synchronization with an internal library, the appropriate listings were generated and filed. Theoretically, the internal and external libraries should be under librarian control and remain synchroneous. However, programmers frequently worked variable hours to accommodate computer availability. When such a situation occurs, a single librarian is not able to maintain complete control and must rely on such techniques as described.

The integration library was maintained under the direction of the technical leader. Programmers were not permitted to update the integration library but informed the technical leader when code was sufficiently tested and ready to be added to the integration library. The internal and external library formats and the librarian's duties were similar to those for the development libraries.

2.2.7 Chief Programmer Team

The PSL project did not use a Chief Programmer as such. Instead, the technical leader was the lead designer, programmed a large percentage of the most complex functions, had technical cognizance over the complete PSL, trained less experienced programmers, and participated in design and code reviews. The major difference was that significant portions of design and coding were performed by other team members.

Section 3

PSL CAPABILITIES

The PSL system can be used to support a varied set of programmer needs, ranging from a small program to large and complex systems. The user has an extensive selection of options, for which default values are provided where practical. Thus a programmer may begin to use the system in a simple straightforward manner and gradually enlist the aid of the more specialized services of the system. A general flowchart is shown in Figure 1.

3.1 System Organization

The PSL system is designed to support the program development and maintenance effort of an entire organization. In a large organization, the system must support many persons working on different programming projects which may be completely unrelated. The PSL provides means of maintaining control over all the data related to each project.

One means of control is the convention used for identifying and organizing the data (source code, compiled modules, test data, etc.) stored under a project. This convention subdivides the data, beginning with the programming project level and proceeding down to single logical units of data. The hierarchical structure of a project is shown in Figure 2.

There are four levels of data identification under the PSL. The highest level of identification is a project. This corresponds to a major programming operation and consists of all of the data related to that operation. Under the WWMCCS PSL system, a project is equivalent to a User Master Catalog (UMC).

The next level of identification is a library. Each project is composed of one or more libraries. Multiple libraries can be used effectively to provide version control over the programming process and to support top-down program development and integration. Any number of libraries may exist under a given project.

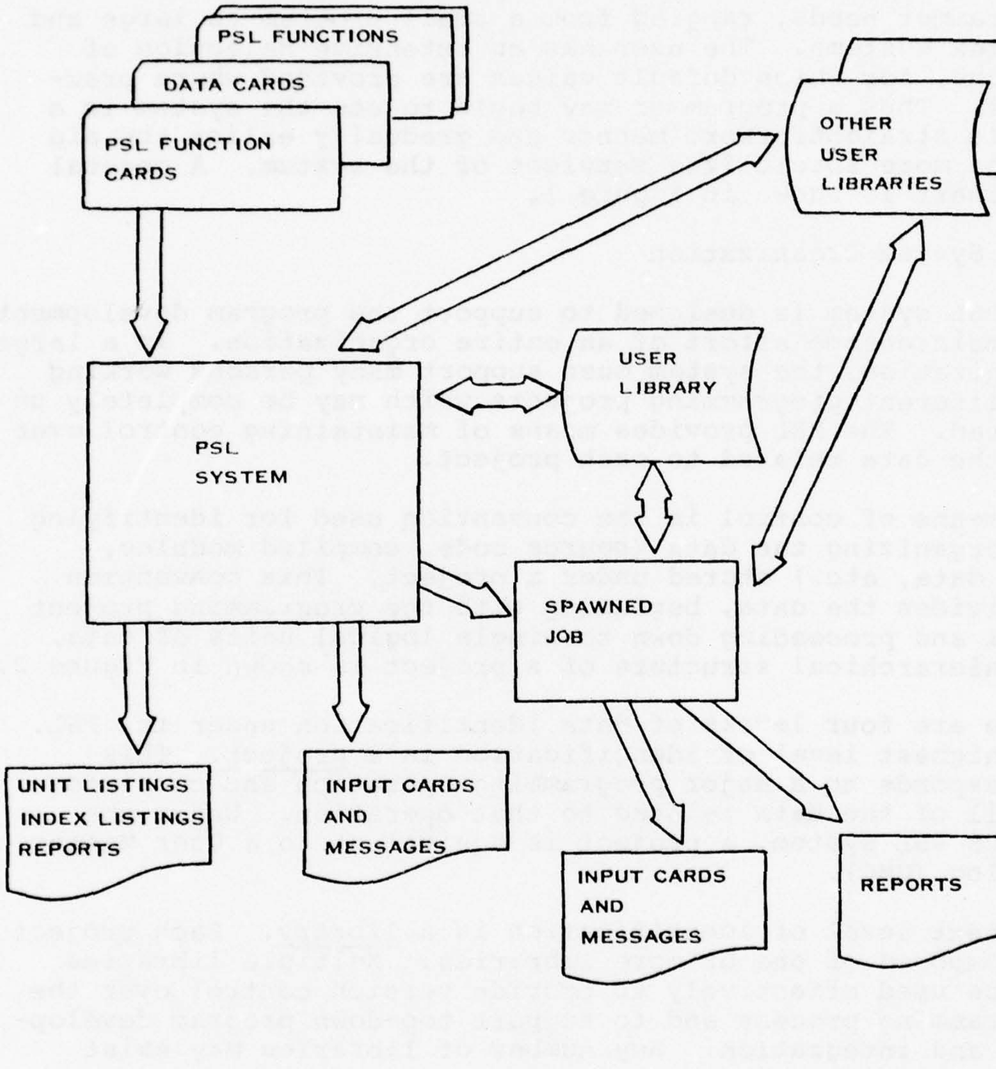
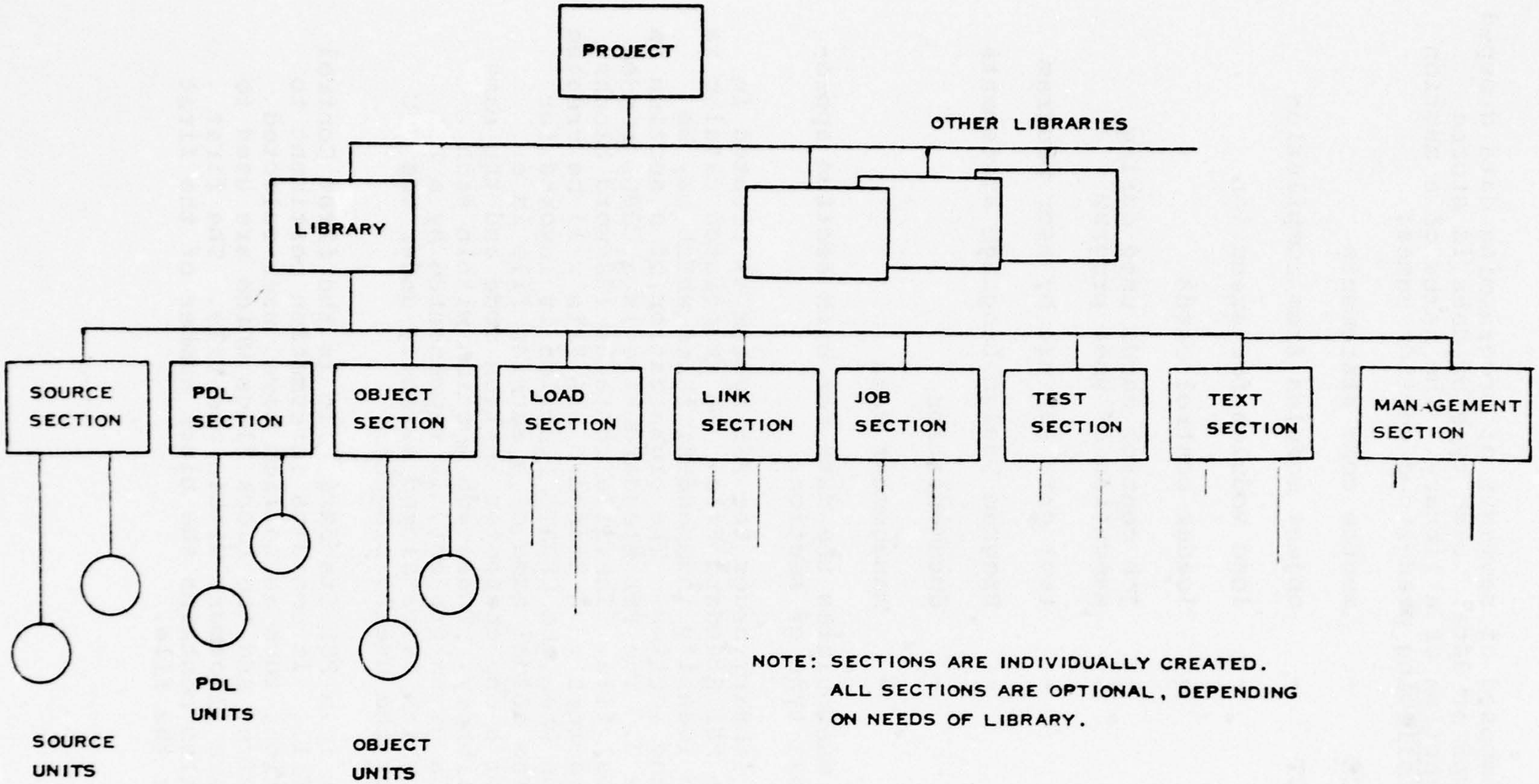


Figure 1. PSL System Flowchart



NOTE: SECTIONS ARE INDIVIDUALLY CREATED.
ALL SECTIONS ARE OPTIONAL, DEPENDING
ON NEEDS OF LIBRARY.

Figure 2. Hierarchical Structure of a Project

A library is composed of segments of programming data grouped according to type of data. Each type of data is stored in a specific section of a library. The names of a section is one of the following predefined section names:

a. SOURCE - source code statements

b. OBJECT - object modules from compilation

c. LOAD - load modules for execution

d. LINK - loader control cards

e. JOB - job control cards used during execution of user program

f. TEST - test data for use by user program

g. PDL - Program Design Language statements

h. TEXT - documentation

i. MGMT - management data

The PSL system manipulates the data for each section appropriately for that type of section.

A section in a library under the PSL system is created in the format of a PSL Standard File. An associated catalog is also created to identify independent files which may be created under the section. The organization of a section is shown in Figure 3. The PSL Standard File is a COBOL random (relative access) file. The file contains 128-word blocks (768 6-bit characters). A Standard PSL File will be created for each section when the CREATE function is invoked for the section. The actual name of a section file is a concatenation of a one-character section code and the name of the user's library. Thus each section within each library, under a given project, is represented by a PSL Standard File, which is cataloged directly under the UMC which represents the user's project.

The first block in a PSL Standard File is the first Control Block for the file. It contains information pertinent to the entire section, such as section name, user-selected options, file size, and the block flags which are used to control the space allotment within the file. The first Control Block also contains the block number of the first Index Block for the file.

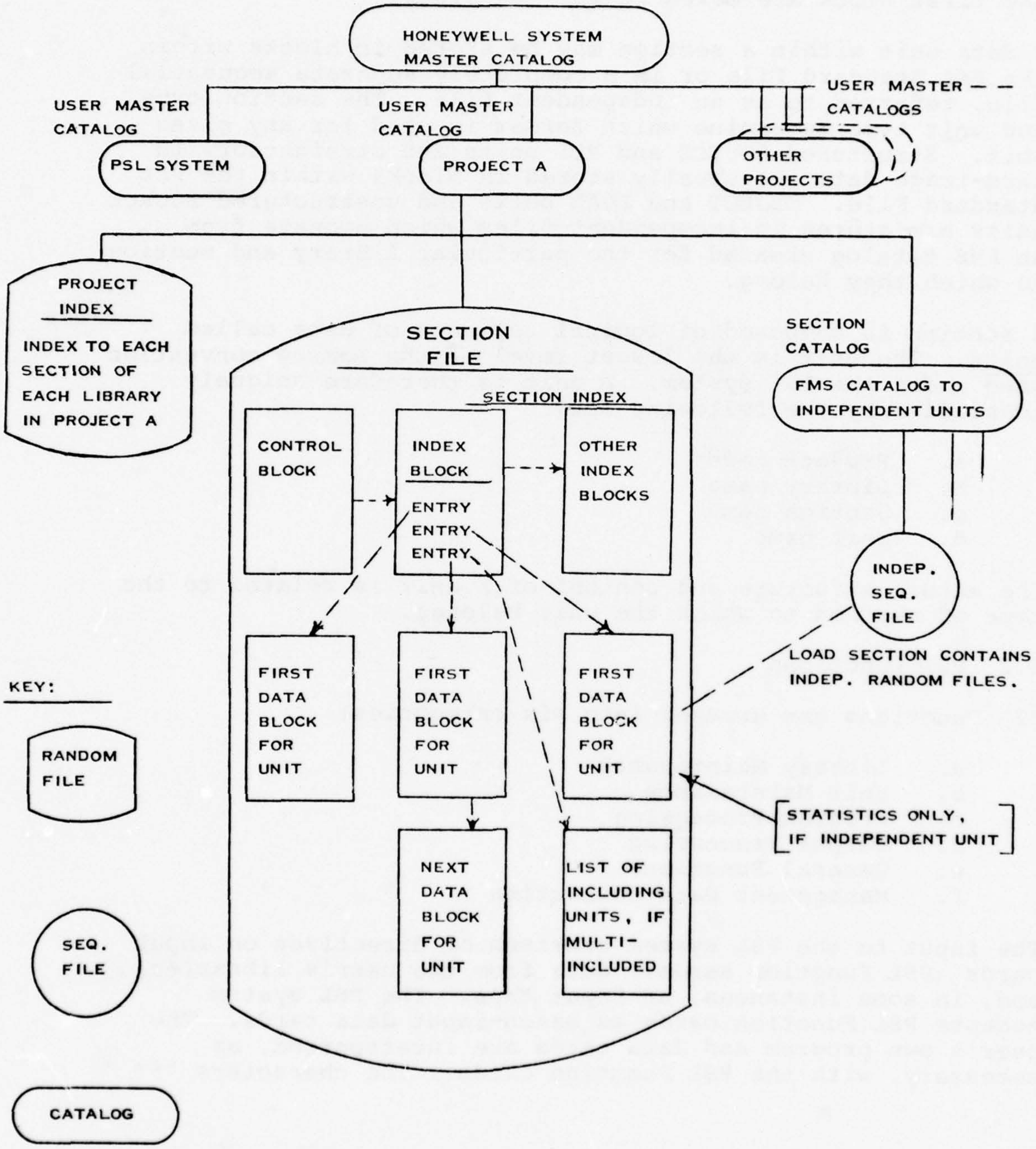

HONEYWELL SYSTEM MASTER CATALOG
USER MASTER CATALOG
USER MASTER CATALOG
USER MASTER CATALOGS
PSL SYSTEM
PROJECT A
OTHER PROJECTS
PROJECT INDEX
INDEX TO EACH SECTION OF EACH LIBRARY IN PROJECT A
SECTION FILE
SECTION INDEX
SECTION CATALOG
FMS CATALOG TO INDEPENDENT UNITS
CONTROL BLOCK
INDEX BLOCK
ENTRY
ENTRY
ENTRY
OTHER INDEX BLOCKS
INDEP. SEQ. FILE
LOAD SECTION CONTAINS INDEP. RANDOM FILES.
FIRST DATA BLOCK FOR UNIT
FIRST DATA BLOCK FOR UNIT
FIRST DATA BLOCK FOR UNIT
STATISTICS ONLY, IF INDEPENDENT UNIT
NEXT DATA BLOCK FOR UNIT
LIST OF INCLUDING UNITS, IF MULTI-INCLUDED
KEY:
RANDOM FILE
SEQ. FILE
CATALOG

Figure 3. Organization of a PSL Section

The Index Blocks contain entries pointing to the location of each data unit within the section. Initially, only one block is assigned to the index. When that block is full, another Index Block is developed and half of the entries from the first block are moved to the new block.

A data unit within a section may be stored in blocks within the PSL Standard File or in a completely separate sequential file, referred to as an independent file. The section type and unit type determine which format is used for any given unit. Structured SOURCE and PDL units and straightforward card-image data are usually stored in blocks within the PSL Standard File. OBJECT and LOAD units and unstructured SOURCE units are stored in independent files which emanate from an FMS catalog created for the particular library and section to which they belong.

A section is composed of logical segments of data called units. The unit is the lowest level of the naming convention used under the PSL system. A unit is therefore uniquely identified by the following set:

a. Project name
b. Library name
c. Section name
d. Unit name

The actual structure and content of a unit is related to the type of section to which the unit belongs.

3.2 PSL Functions

PSL Functions are grouped into six categories:

a. Library Maintenance
b. Unit Maintenance
c. Program Processing
d. Output Processing
e. General Functions
f. Management Data Collection

The input to the PSL system consists of directives on input cards (PSL Function cards), data from the user's libraries, and, in some instances, an input tape. The PSL system accepts PSL Function cards as batch-input data cards. The user's own program and data cards are interspersed, as necessary, with the PSL Function cards. The characters "** " in columns 1 through 3 identify a card as a PSL Function card or a Function continuation card. The PSL cards contain the name of a Function requested and, if appropriate, sets of keywords and value-entries. Figure 4 lists the PSL Functions, and associated keyworls. The functions under each of the six categories are described in the following paragraphs.

3.2.1 Library Maintenance

Before programs are developed under the PSL system, a user establishes his project and library. The following PSL Functions are used for general library maintenance.

3.2.1.1 INITIAL - Initialize a Project

Under the PSL system, a user stores programs and data in discrete units, within sections of a library, under a project. A project must be initialized before libraries are built under it. The name of the User Master Catalog (UMC), which was assigned to the project by the WWMCCS installation, is used as the name of the project. The PSL Function INITIAL establishes the project as a PSL project, prepares an index for library-sections under the project, and stores the password associated with the USERID.

3.2.1.2 CREATE - Create a Section

After a project is initialized, the CREATE Function is used to build sections in the user's library. The library name uniquely identifies a group of standard sections under a project.

Within a library, the PSL system allows the user to create any of the eight standard PSL sections:

a. SOURCE - This section contains all of the source code which is stored in a library, regardless of language or structure. Unit of structured code, which at present include Structured COBOL (SCOBOL) and Structured FORTRAN (SPFORT) are stored in blocks in a random file. It is recommended that the size of structured units be restricted to one page of code. For compilation, structured units are combined in accordance with the INCLUDE statements found in the units. Units of unstructured code (GMAP, FORTRAN, and COBOL) are stored as separately compilable units in sequential files.

Functions

- ADD
- AUTHOR
- BACKUP
- CHANGE
- COMPILE
- CREATE
- CSCAN
- DOCUMENT
- EXECUTE
- INDEX
- INITIAL
- JCL
- LINK
- MDCOLLECT
- MDFORMAT
- MDPLAN
- MDPRINT
- MDUPDATE
- MDXCHECK
- MOVE
- PARAM
- PURGE
- REPLACE
- RESTORE
- SOURCE
- TERMINATE

Subfunctions

- COPY
- DELETE
- EJECT
- HEADER
- INSERT
- MODIFY
- SHIFT
- SPACE
- TEXT

Keywords

PSL

- AFTER
- ALL
- ARCHIVE
- BACKUP
- CALL
- CLASS
- COLUMN
- COMPRESS
- CYCLE
- ELEMENT
- FMS
- FROM
- HISTORY
- HPRINT
- HSPACE
- INDENT
- JOB
- KEY
- LADJUST
- LANGUAGE
- LEVEL
- LIB1
- .
- .
- .
- LIB9
- LIBRARY
- LINE(S)
- LINK
- LOAD
- LOADER
- LSPACE
- MEDIA
- MGMTDATA
- MOD
- MODULE
- NBRLINES
- NEST
- NEWPASS
- OBJECT
- OLDLIB
- OLDPROJ
- OLDSEC
- OLDUNIT

- OP
- OPGMR
- OPTION
- PAGE
- PARAMCARD
- PASSWORD
- PGMR
- PLINES
- PROCEDURE
- PROJ1
- .
- .
- .
- PROJ9
- PROJECT
- RECYCLE
- REPLACE
- REPORT
- SECTION
- SPCHECK
- SPLENGTH
- STANDARD
- START
- STRING
- SYSTEM
- SUBSYSTEM
- TO
- UNIT
- UPASS
- UPGMR
- USPACE
- UTYPE

FMS

- ABORT
- ACCESS
- APPEND
- AUDIT
- BLOCKS
- CREATE
- DELETE
- DEVICE
- EXCLUDE
- EXECUTE
- FCLASS
- INCRSAVE
- LINKS
- LLINKS
- LOCK
- MODIFY
- PAGESIZE
- PURGE
- RDERR
- READ
- RECOVERY
- RESET
- SIZE
- VERIFY
- WRITE

Figure 4. Functions, Subfunctions, and Keywords

b. OBJECT - The object modules which result from compilation of source code are stored in the OBJECT section. The name of the object unit is the same as the name of the top-most unit of source code. Object modules are used singly or in combination to form a load module for execution.

c. LOAD - If the load module is to be saved on permanent storage, it is stored as a system-loadable program in the LOAD section. Units in the OBJECT and LOAD sections are produced by HIS system functions and are therefore not maintained with unit maintenance Functions, as are the other sections. A load module is normally produced by the General Loader, using one or more object modules. However, the System Library Editor may be selected by a keyword on the LINK Function card.

d. LINK - Units in this section are used to store control cards for directing the loading process when more than one object module is to be loaded.

e. JOB - Control cards may be stored in units in the JOB section and invoked by the EXECUTE Function. The cards will be added to the program execution activity to provide the job control cards the user's program requires.

f. TEST - This is a general section for card-format data usually used for test input.

g. PDL - Program Design Language statements are stored in units in the PDL section. PDL consists of English-like statements which follow the basic rules of structured programming and are used to define the program structure and logic.

h. TEXT - This section contains units of standard textual material, which are written primarily for use as program documentation. The units may be maintained under the PSL system and printed as any other card-format unit.

i. MGMT - A management plan is initialized in the created MGMT section and subsequently updated to define the project elements that comprise the management data report requirement. Format units are established to define the report contents and input units are added to introduce manual data. Both automatic data from unit accounting records and manually input data are collected and archived in collection units for subsequent input to management data reports.

3.2.1.3 BACKUP

The PSL system provides a BACKUP Function to save the user's programs and data on tape. The BACKUP may be invoked for a complete project, a library, or a section.

3.2.1.4 RESTORE

The RESTORE Function is used to write the contents of the BACKUP tape into the library. This Function may be used to restore the complete contents of the BACKUP tape, or it may selectively recover a portion of the total data tape at as low a level as a unit of code.

3.2.1.5 TERMINATE

This Function enables a user to delete a section, a library, or an entire project. Files which are released are overwritten.

3.2.2 Unit Maintenance

After a user has initialized a project and has created the sections which are needed in his library, data may be stored in these units. The user may add and update data in those sections which are used for card-image data (SOURCE, PDL, LINK, JOB, TEST, and TEXT). Units in the OBJECT and LOAD sections are not maintained directly by the user, since they are automatically created and updated by the PSL system as a result of the COMPILE and LINK Functions (see subsection 3.1.3). MGMT units are updated by the management data collection Functions.

The PSL system automatically maintains accounting information for each unit in each section. This information is initialized when the unit is created and updated when the unit is modified.

The following PSL Functions are used to add, change, and purge units of card-image data in sections (other than MGMT) in the user's library.

3.2.2.1 ADD - Add a Unit

The ADD Function is used for the original introduction of data into a unit. The accounting information for the unit is initialized by the ADD Function. The actual data cards for the new unit follow the ADD Function card in the run stream. After the ADD is completed, a listing of the unit will be automatically generated.

3.2.2.2 REPLACE - Replace a Unit

After data has been added to a unit, the REPLACE Function is used to completely replace all of the data lines in the unit. Accounting information is not re-initialized, but is updated.

3.2.2.3 CHANGE - Change a Unit

The CHANGE Function is used to modify the contents of a unit. Modification details are provided on Subfunction cards (COPY, DELETE, INSERT, MODIFY, and SHIFT) which follow the CHANGE card. New source statements follow the INSERT and MODIFY Subfunction cards. After the CHANGE operation is completed, a listing of the altered unit will be automatically generated.

3.2.2.4 MOVE - Move a Unit

Units are moved from one library to another, or within a library, with the MOVE Function.

3.2.2.5 PURGE - Purge a Unit

An individual unit is removed from a library with the PURGE Function. If the PURGE of a unit results in the release of a file, the file contents will be overwritten.

3.2.3 Program Processing

The PSL system provides many facilities to support the compilation, loading, and execution of programs. One of the more powerful capabilities available for these Functions is the optional search through multiple libraries during retrieval.

3.2.3.1 COMPILE - Compile a Module

The COMPILE Function retrieves units from the SOURCE section, invokes the appropriate precompiler for structed source code, compiles (or assembles) the resulting stream of code, and stores the compiler product in a unit in the OBJECT section. The UNIT name is the name of the top-most source unit of the module which is to be compiled. This name will be used for the name of the compiled OBJECT unit. The processing which is invoked by the COMPILE Function depends upon the language of the unit. Under the present PSL system, procedures exist to process languages COBOL, SCOBOL, (Structured COBOL), FORTRAN, SPFORT (Structured FORTRAN), FORTY, JOVIAL, and GMAP.

3.2.3.2 LINK - Link a Program

The LINK Function retrieves one or more object modules from the OBJECT section, loads the object modules, and retains a permanent copy of the system-loadable file. The resulting load module is stored as a unit in the LOAD section.

If more than one object module is to be linked together to form the load module, loader control cards are retrieved from a unit in the LINK section.

3.2.3.3 EXECUTE - Execute a Program

Job control cards for execution of a program are previously stored as a unit in the JOB section. The EXECUTE Function invokes the execution of a program using the indicated job control cards as the input run stream.

The program itself may be retrieved by the PSL system in one of the following forms:

a. A load module from the LOAD section

b. An object module from the OBJECT section which will be processed by the Loader

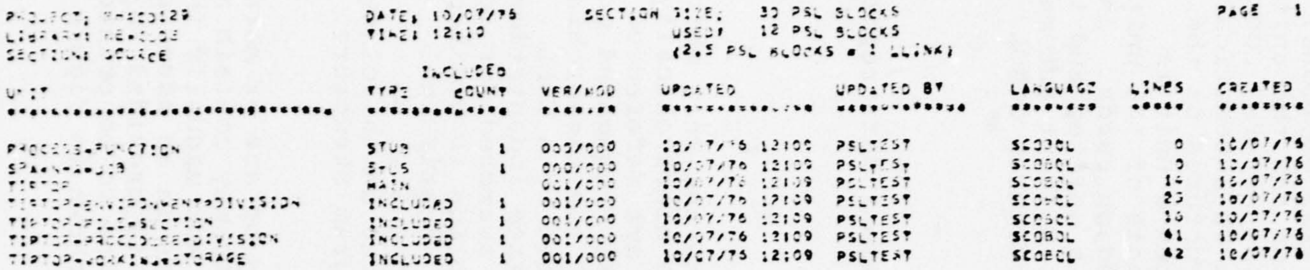
c. A series of object modules which are selected as a result of Loader control cards in a unit of the LINK section and which will be processed by the Loader.

3.2.4 Output Processing

Six PSL Functions are available in the PSL system to obtain reports.

3.2.4.1 INDEX - Print a Section Index

The INDEX Function prints a status report for a section. The report contains information pertaining to the section as a whole, as well as a listing of the index for the section. An example of a Section Index Report is shown in Figure 5.



```
PROJECT: MHSD129                     DATE: 10/07/76     SECTION SIZE:   30 PSL BLOCKS                               PAGE   1
LIBRARY: NEWCODE                     TIME: 12:10                USED:   12 PSL BLOCKS
SECTION: SOURCE                                                 (2.5 PSL BLOCKS = 1 LLINK)
                                            INCLUDED
UNIT                                 TYPE      COUNT   VER/MOD    UPDATED          UPDATED BY       LANGUAGE   LINES   CREATED
*******************************      *************     *******    **************   ************     ********   *****   ********

PROCESS-FUNCTION                     STUB          1   000/000    10/07/76 12:09   PSLTEST          SCOBOL         0   10/07/76
SPAWN-AWU19                          STUB          1   000/000    10/07/76 12:09   PSLTEST          SCOBOL         0   10/07/76
TIPTOP                               MAIN              001/000    10/07/76 12:09   PSLTEST          SCOBOL        14   10/07/76
TIPTOP-ENVIRONMENT-DIVISION          INCLUDED      1   001/000    10/07/76 12:09   PSLTEST          SCOBOL        23   10/07/76
TIPTOP-FILE-SECTION                  INCLUDED      1   001/000    10/07/76 12:09   PSLTEST          SCOBOL        16   10/07/76
TIPTOP-PROCEDURE-DIVISION            INCLUDED      1   001/000    10/07/76 12:09   PSLTEST          SCOBOL        41   10/07/76
TIPTOP-WORKING-STORAGE               INCLUDED      1   001/000    10/07/76 12:09   PSLTEXT          SCOBOL        42   10/07/76
```

Figure 5. Section Index Report

3.2.4.2 SOURCE - Print a Card-Image Unit

The SOURCE Function can be used to print a listing of any unit which is in card-image format. This includes the units in the SOURCE, PDL, LINK, JOB, MGMT, TEST, and TEXT sections. If the unit is from the SOURCE or the PDL section and the unit is written in a supported structured language (Structured COBOL, and Structured FORTRAN), the structured code is automatically indented for the proper alignment of the structures on the listing. This listing is always provided when a unit is added to the library or modified. The contents of the unit may, on option, be written to tape or punched on cards. In these latter cases, the header information is omitted and the code is reproduced as it exists in the unit, without automatic indentation. An example of a listing of structured COBOL (SCOBOL) is shown in Figure 6.

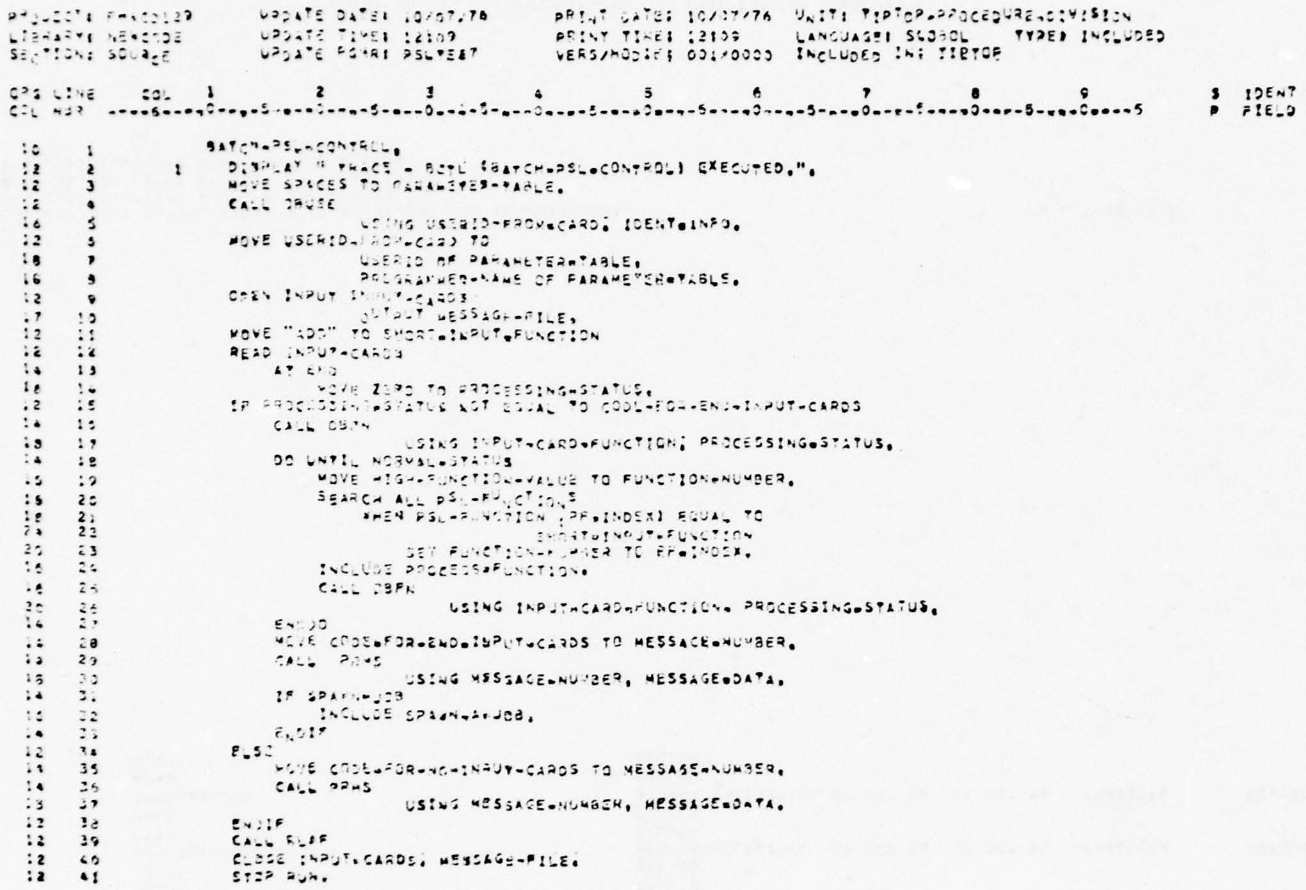
3.2.4.3 MDPRINT - Print Reports

The MDPRINT Function is used to print management data reports. There are two categories of such reports:

a. Program Structure Report
b. Management Data Report

The Program Structure (PS) report begins with the unit which is named on the Function card and develops a hierarchically nested and indented list of all units which are referenced by an INCLUDE statement, either in the originally-requested unit or in units which are themselves INCLUDEd in the hierarchical program structure. Units which are referenced by a CALL statement are also printed with the appropriate indentation in the list, but they are not automatically searched for lower levels of INCLUDE or CALL statements. An option is available to invoke a PS report for all CALLed units. Statistical organization information is printed for each unit. An alphabetically-arranged list of all referenced units follows the hierarchical list. An example of a Program Structure Report is shown in Figure 7.

A Management Data report prints the contents of one or more management data units. Management data units may contain a combination of automatically collected data and manually input data combined according to the specifications in an associated user-defined format unit. A general format is provided to accommodate a wide variety of data collections to be printed. A representative Management Data Report is shown in Figure 8.

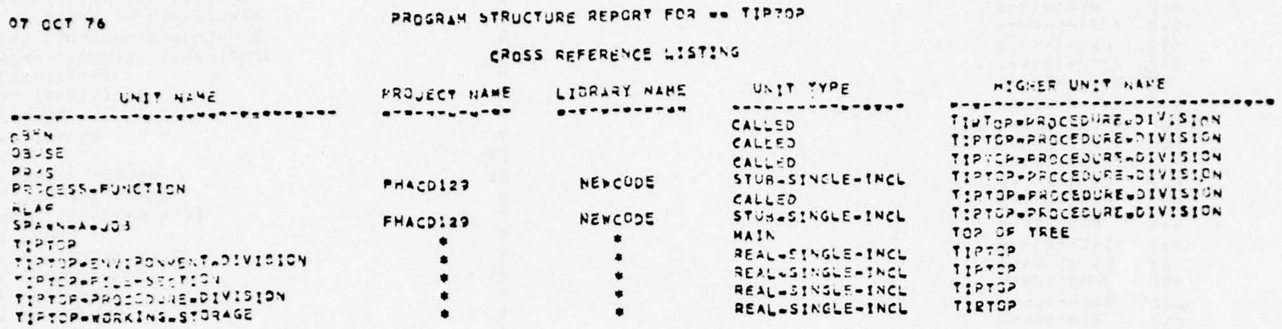


```
PROJECT: FM400129      UPDATE DATE: 10/07/76      PRINT DATE: 10/27/76  UNIT: TIPTOP-PROCEDURE-DIVISION
LIBRARY: NEWCODE       UPDATE TIME: 12:09         PRINT TIME: 12:09     LANGUAGE: SCOBOL     TYPE: INCLUDED
SECTION: SOURCE        UPDATE PGMR: PSLTEST       VERS/MODIF: 001/0000  INCLUDED IN: TIPTOP

OPS LINE     COL  1         2         3         4         5         6         7         8         9            $ IDENT
COL NBR  ----6----0----5----0----5----0----5----0----5----0----5----0----5----0----5----0----5----0----5     P FIELD

 10    1            BATCH-PSL-CONTROL.
 12    2       1      DISPLAY " TRACE - BCTL (BATCH-PSL-CONTROL) EXECUTED.".
 12    3              MOVE SPACES TO PARAMETER-TABLE.
 12    4              CALL DRUSE
 16    5                          USING USERID-FROM-CARD, IDENT-INFO.
 12    6              MOVE USERID-FROM-CARD TO
 18    7                          USERID OF PARAMETER-TABLE,
 16    8                          PROGRAMMER-NAME OF PARAMETER-TABLE.
 12    9              OPEN INPUT INPUT-CARDS
 17   10                          OUTPUT MESSAGE-FILE.
 12   11              MOVE "ADD" TO SHORT-INPUT-FUNCTION
 12   12              READ INPUT-CARDS
 14   13                  AT END
 16   14                      MOVE ZERO TO PROCESSING-STATUS.
 12   15              IF PROCESSING-STATUS NOT EQUAL TO CODE-FOR-END-INPUT-CARDS
 14   16                  CALL DBIN
 18   17                              USING INPUT-CARD-FUNCTION, PROCESSING-STATUS.
 14   18                  DO UNTIL NORMAL-STATUS
 15   19                      MOVE HIGH-FUNCTION-VALUE TO FUNCTION-NUMBER.
 15   20                      SEARCH ALL PSL-FUNCTIONS
 18   21                          WHEN PSL-FUNCTION (PF-INDEX) EQUAL TO
 24   22                                          SHORT-INPUT-FUNCTION
 20   23                              SET FUNCTION-NUMBER TO PF-INDEX.
 16   24                      INCLUDE PROCESS-FUNCTION.
 16   25                      CALL DBFN
 20   26                                  USING INPUT-CARD-FUNCTION, PROCESSING-STATUS.
 14   27                  ENDDO
 14   28                  MOVE CODE-FOR-END-INPUT-CARDS TO MESSAGE-NUMBER.
 14   29                  CALL  PRMS
 16   30                              USING MESSAGE-NUMBER, MESSAGE-DATA.
 14   31                  IF SPAWN-JOB
 16   32                      INCLUDE SPAWN-A-JOB.
 14   33                  ENDIF
 12   34              ELSE
 14   35                  MOVE CODE-FOR-NO-INPUT-CARDS TO MESSAGE-NUMBER.
 14   36                  CALL PRMS
 13   37                              USING MESSAGE-NUMBER, MESSAGE-DATA.
 12   38              ENDIF
 12   39              CALL RLSF
 12   40              CLOSE INPUT-CARDS, MESSAGE-FILE.
 12   41              STOP RUN.
```

Figure 6. Structured COBOL (SCOBOL)

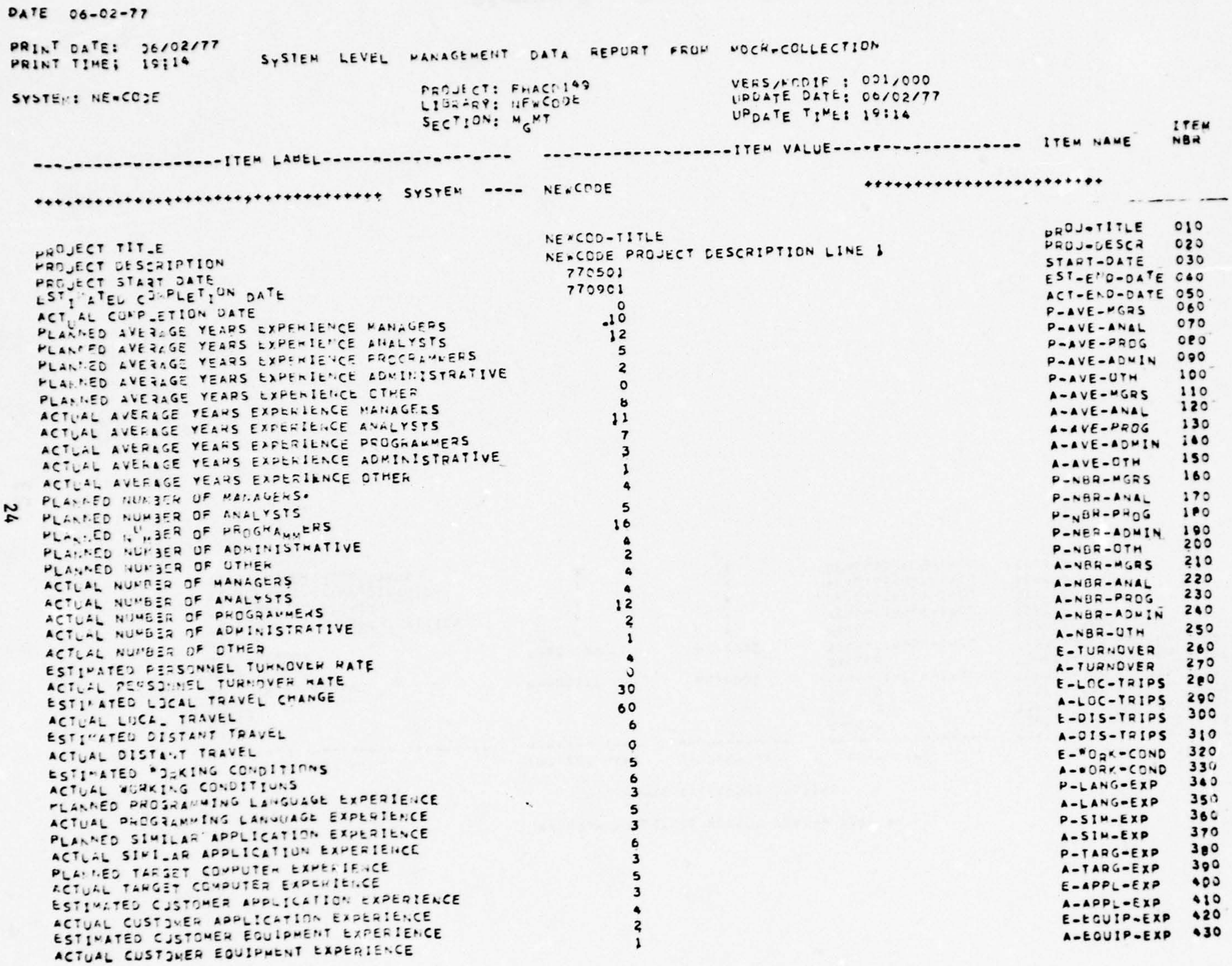
```
07 OCT 76                          PROGRAM STRUCTURE REPORT FOR -- TIPTOP

MAXIMUM NBR. OF LEVELS TO BE PRINTED = 50                                                                          CALL OPTION = NO

UNIT  UNIT                                                                 ORIGINATE    LAST        PROJECT      LIBRARY    SP
LEVEL LINES       UNIT NAME                             UNIT TYPE            DATE       UPDATE       NAME         NAME     FLAG
----- ----  ------------------------------------  ----------------  ---------  ---------  ------------  --------  ----
  1     16  TIPTOP                                 MAIN              07 OCT 76  07 OCT 75  FHACD129      NEWCODE
  2     23    TIPTOP-ENVIRONMENT-DIVISION          REAL-SINGLE-INCL  07 OCT 76  07 OCT 76      *             *
  2     16    TIPTOP-FILE-SECTION                  REAL-SINGLE-INCL  07 OCT 76  07 OCT 76      *             *
  2     62    TIPTOP-WORKING-STORAGE               REAL-SINGLE-INCL  07 OCT 76  07 OCT 76      *             *
  2     41    TIPTOP-PROCEDURE-DIVISION            REAL-SINGLE-INCL  07 OCT 76  07 OCT 76      *             *
  3             DBUSE                              CALLED
  3             DBEN                               CALLED
  3             PROCESS-FUNCTION                   STUB-SINGLE-INCL  07 OCT 76  07 OCT 75  FHACD129      NEWCODE
  3             DBEN                               CALLED
  3             PRMS                               CALLED
  3             SPAWN-A-JOB                        STUB-SINGLE-INCL  07 OCT 76  07 OCT 76  FHACD129      NEWCODE
  3             PRMS                               CALLED
  3             RLAP                               CALLED

SP-FLAG-MESSAGES (NBRS. UNDER SP-FLAG HEADER ARE CUMULATIVE);                                         * - SAME NAME
    1 - UNIT CONTAINS GO TO- STATEMENTS
    2 - NBR. OF LINES EXCEEDS DEFINED LIMITS
    4 - MORE THAN ONE STATEMENT ON A LINE
```

Figure 7. Program Structure Report
(Part 1 of 2)



```
07 OCT 76                                PROGRAM STRUCTURE REPORT FOR -- TIPTOP

                                               CROSS REFERENCE LISTING

            UNIT NAME                    PROJECT NAME    LIBRARY NAME      UNIT TYPE                HIGHER UNIT NAME
------------------------------------     ------------    ------------   ----------------   --------------------------------
OPEN                                                                    CALLED             TIPTOP-PROCEDURE-DIVISION
OBUSE                                                                   CALLED             TIPTOP-PROCEDURE-DIVISION
PO-S                                                                    CALLED             TIPTOP-PROCEDURE-DIVISION
PROCESS-FUNCTION                          PHACD129         NEWCODE      STUB-SINGLE-INCL   TIPTOP-PROCEDURE-DIVISION
RLAF                                                                    CALLED             TIPTOP-PROCEDURE-DIVISION
SPAWN-A-JOB                               FHACD129         NEWCODE      STUB-SINGLE-INCL   TIPTOP-PROCEDURE-DIVISION
TIPTOP                                        *               *         MAIN               TOP OF TREE
TIPTOP-ENVIRONMENT-DIVISION                   *               *         REAL-SINGLE-INCL   TIPTOP
TIPTOP-FILE-SECTION                           *               *         REAL-SINGLE-INCL   TIPTOP
TIPTOP-PROCEDURE-DIVISION                     *               *         REAL-SINGLE-INCL   TIPTOP
TIPTOP-WORKING-STORAGE                        *               *         REAL-SINGLE-INCL   TIRTOP
```

Figure 7. Program Structure Report
(Part 2 of 2)



```
DATE  06-02-77

PRINT DATE:  06/02/77
PRINT TIME;  19:14        SYSTEM  LEVEL  MANAGEMENT  DATA  REPORT  FROM  MOCK-COLLECTION

SYSTEM: NEWCODE                         PROJECT: FHACD149          VERS/MODIF : 001/000
                                        LIBRARY: NEWCODE           UPDATE DATE: 06/02/77
                                        SECTION: MGMT              UPDATE TIME: 19:14
                                                                                                        ITEM
---.-------------ITEM LABEL-----------------   ------------------ITEM VALUE-------------------  ITEM NAME    NBR

+++++++++++++++++++++++++++++++++++  SYSTEM  ----  NEWCODE                  ++++++++++++++++++++++
                                                                                              ------  ---
PROJECT TITLE                                NEWCOD-TITLE                                    PROJ-TITLE    010
PROJECT DESCRIPTION                          NEWCODE PROJECT DESCRIPTION LINE 1              PROJ-DESCR    020
PROJECT START DATE                             770501                                        START-DATE    030
ESTIMATED COMPLETION DATE                      770901                                        EST-END-DATE  040
ACTUAL COMPLETION DATE                              0                                        ACT-END-DATE  050
PLANNED AVERAGE YEARS EXPERIENCE MANAGERS          10                                        P-AVE-MGRS    060
PLANNED AVERAGE YEARS EXPERIENCE ANALYSTS          12                                        P-AVE-ANAL    070
PLANNED AVERAGE YEARS EXPERIENCE PROGRAMMERS        5                                        P-AVE-PROG    080
PLANNED AVERAGE YEARS EXPERIENCE ADMINISTRATIVE     2                                        P-AVE-ADMIN   090
PLANNED AVERAGE YEARS EXPERIENCE OTHER              0                                        P-AVE-OTH     100
ACTUAL AVERAGE YEARS EXPERIENCE MANAGERS            8                                        A-AVE-MGRS    110
ACTUAL AVERAGE YEARS EXPERIENCE ANALYSTS           11                                        A-AVE-ANAL    120
ACTUAL AVERAGE YEARS EXPERIENCE PROGRAMMERS         7                                        A-AVE-PROG    130
ACTUAL AVERAGE YEARS EXPERIENCE ADMINISTRATIVE      3                                        A-AVE-ADMIN   140
ACTUAL AVERAGE YEARS EXPERIENCE OTHER               1                                        A-AVE-OTH     150
PLANNED NUMBER OF MANAGERS.                         4                                        P-NBR-MGRS    160
PLANNED NUMBER OF ANALYSTS                          5                                        P-NBR-ANAL    170
PLANNED NUMBER OF PROGRAMMERS                      16                                        P-NBR-PROG    180
PLANNED NUMBER OF ADMINISTRATIVE                    4                                        P-NBR-ADMIN   190
PLANNED NUMBER OF OTHER                             2                                        P-NBR-OTH     200
ACTUAL NUMBER OF MANAGERS                           4                                        A-NBR-MGRS    210
ACTUAL NUMBER OF ANALYSTS                           4                                        A-NBR-ANAL    220
ACTUAL NUMBER OF PROGRAMMERS                       12                                        A-NBR-PROG    230
ACTUAL NUMBER OF ADMINISTRATIVE                     2                                        A-NBR-ADMIN   240
ACTUAL NUMBER OF OTHER                              1                                        A-NBR-OTH     250
ESTIMATED PERSONNEL TURNOVER RATE                   4                                        E-TURNOVER    260
ACTUAL PERSONNEL TURNOVER RATE                      0                                        A-TURNOVER    270
ESTIMATED LOCAL TRAVEL CHANGE                      30                                        E-LOC-TRIPS   280
ACTUAL LOCAL TRAVEL                                60                                        A-LOC-TRIPS   290
ESTIMATED DISTANT TRAVEL                            6                                        E-DIS-TRIPS   300
ACTUAL DISTANT TRAVEL                               0                                        A-DIS-TRIPS   310
ESTIMATED WORKING CONDITIONS                        5                                        E-WORK-COND   320
ACTUAL WORKING CONDITIONS                           6                                        A-WORK-COND   330
PLANNED PROGRAMMING LANGUAGE EXPERIENCE             3                                        P-LANG-EXP    340
ACTUAL PROGRAMMING LANGUAGE EXPERIENCE              5                                        A-LANG-EXP    350
PLANNED SIMILAR APPLICATION EXPERIENCE              3                                        P-SIM-EXP     360
ACTUAL SIMILAR APPLICATION EXPERIENCE               6                                        A-SIM-EXP     370
PLANNED TARGET COMPUTER EXPERIENCE                  3                                        P-TARG-EXP    380
ACTUAL TARGET COMPUTER EXPERIENCE                   5                                        A-TARG-EXP    390
ESTIMATED CUSTOMER APPLICATION EXPERIENCE           3                                        E-APPL-EXP    400
ACTUAL CUSTOMER APPLICATION EXPERIENCE              4                                        A-APPL-EXP    410
ESTIMATED CUSTOMER EQUIPMENT EXPERIENCE             2                                        E-EQUIP-EXP   420
ACTUAL CUSTOMER EQUIPMENT EXPERIENCE                1                                        A-EQUIP-EXP   430
```

Figure 8. Management Data Report

3.2.4.4 AUTHOR - Print by Author

The AUTHOR Function can be used to print a list of unit names, or listings of the actual units, which were originally generated by and/or updated by a specific programmer. An example of an AUTHOR Report ir shown in Figure 9.

3.2.4.5 DOCUMENT - Print Text

The DOCUMENT Function is used to print documentation stored in a library in the form of program design language, structured source code, text, etc. Output requirements are specified through keyword options provided on subfunction cards (HEADER, TEXT, EJECT, and SPACE). An example of a DOCUMENT Report is shown in Figure 10.

3.2.4.6 CSCAN - Print by Character String

The CSCAN Function is used to scan all units of the indicated section to locate a specific character string. The string may be up to 48 characters in length. The resulting output will be a list of the unit names containing the specific character string and the corresponding lines of code for each occurrance. Figure 11 contains an example of character scan output.

3.2.5 General Functions

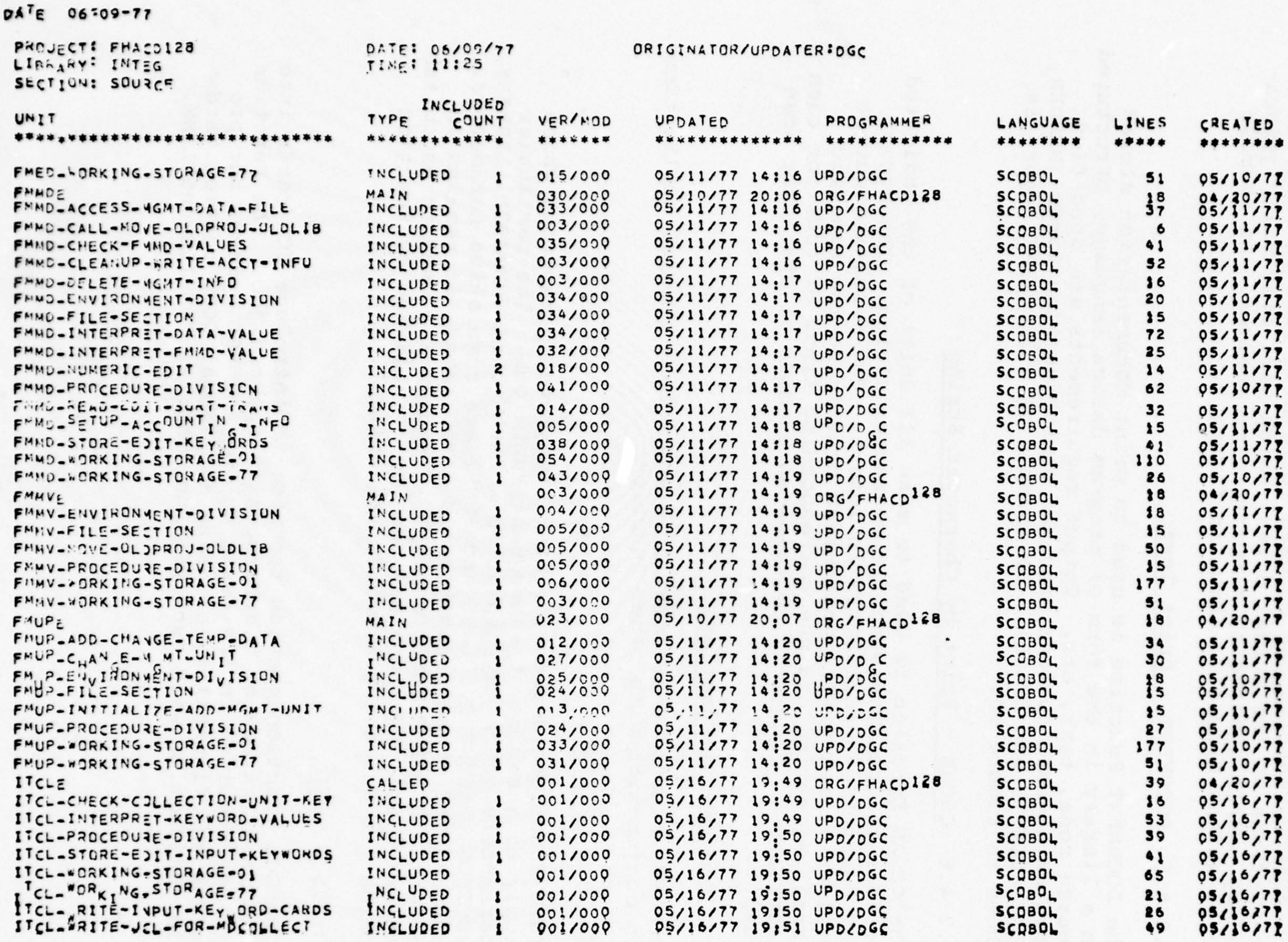
Two PSL Functions are available which have general application in conjunction with other Functions.

3.2.5.1 PARAM

This Function is used to enter high-level parameters which apply to a series of subsequent Functions. The particular parameters which may appear on a PARAM card (PROJECT, LIBRARY, SECTION, PASSWORD, CLASS and PGMR) are cumulative parameters. When any of these parameters is established, it remains in effect. A PARAM Function card is ordinarily used to establish these values, but they may also be established by Function cards for which the keywords are valid.

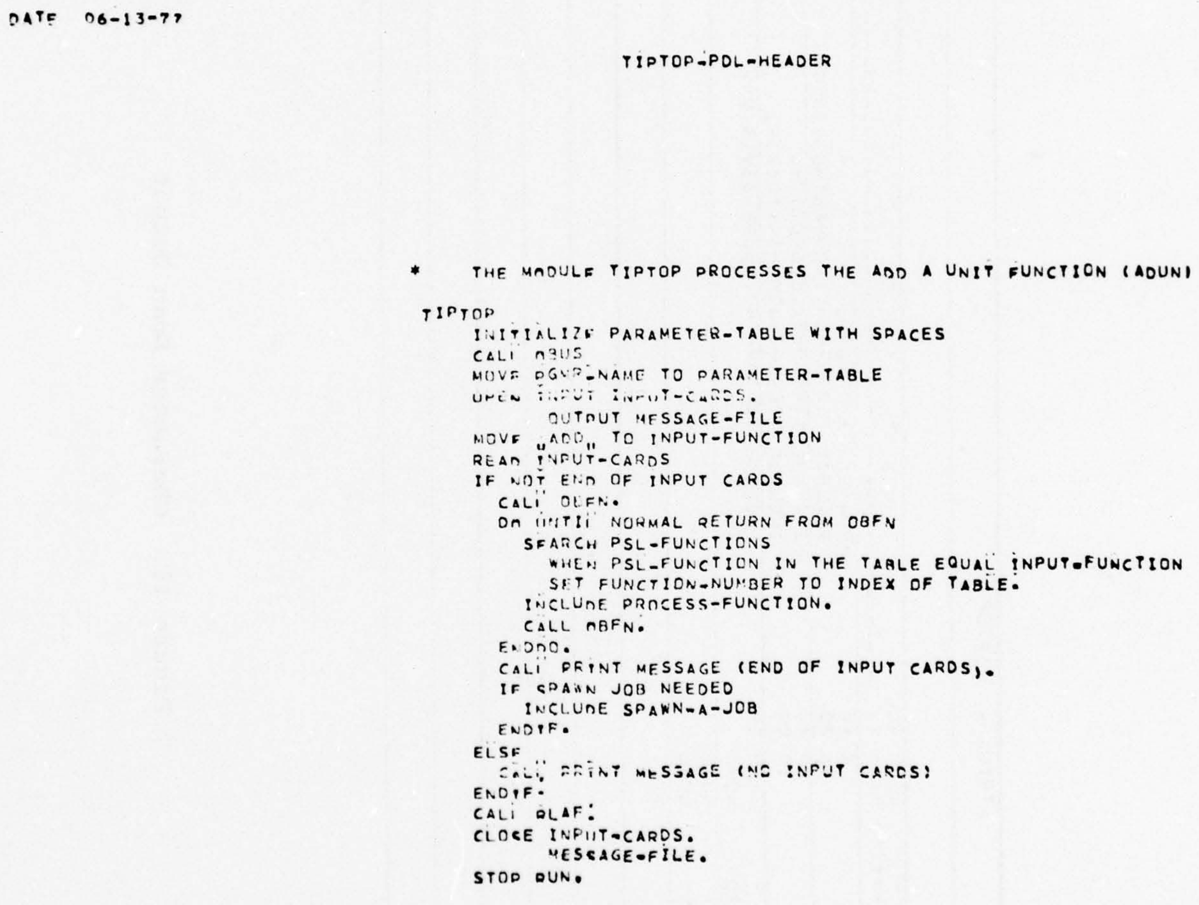
3.2.5.2 JCL

The JCL Function enables the user to introduce JCL cards into the input stream of a subsequently spawned job. The Function is not required in the basic use of the PSL system, but provides the flexibility of using additional GCOS control cards in conjunction with such PSL Functions as COMPILE and LINK.



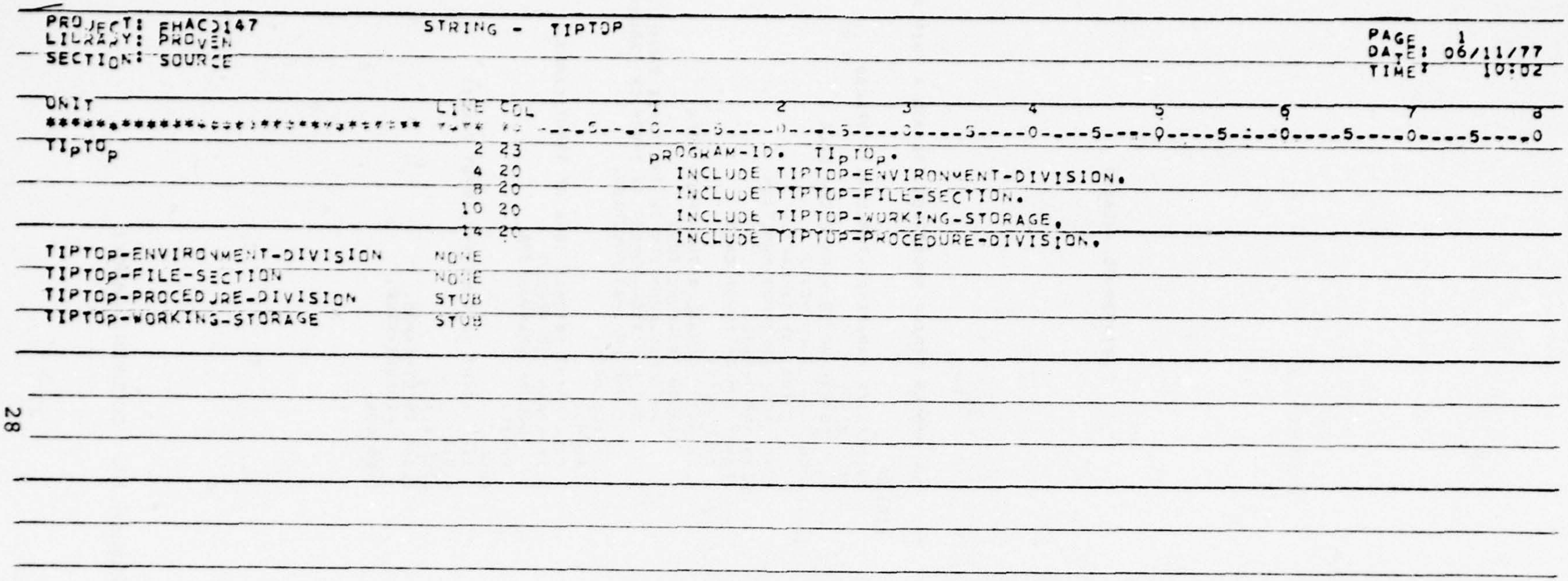
```
DATE 06-09-77

PROJECT: FHACD128          DATE: 06/09/77        ORIGINATOR/UPDATER:DGC
LIBRARY: INTEG             TIME: 11:25
SECTION: SOURCE
                                INCLUDED
UNIT                       TYPE      COUNT  VER/MOD  UPDATED         PROGRAMMER     LANGUAGE  LINES  CREATED
************************** ************* ******* ************** ************  ********  *****  ********

FMED-WORKING-STORAGE-77    INCLUDED   1     015/000  05/11/77 14:16 UPD/DGC        SCOBOL       51  05/10/77
FMMDE                      MAIN             030/000  05/10/77 20:06 ORG/FHACD128   SCOBOL       18  04/20/77
FMMD-ACCESS-MGMT-DATA-FILE INCLUDED   1     033/000  05/11/77 14:16 UPD/DGC        SCOBOL       37  05/11/77
FMMD-CALL-MOVE-OLDPROJ-OLDLIB INCLUDED 1    003/000  05/11/77 14:16 UPD/DGC        SCOBOL        6  05/11/77
FMMD-CHECK-FMMD-VALUES     INCLUDED   1     035/000  05/11/77 14:16 UPD/DGC        SCOBOL       41  05/11/77
FMMD-CLEANUP-WRITE-ACCT-INFO INCLUDED 1     003/000  05/11/77 14:16 UPD/DGC        SCOBOL       52  05/11/77
FMMD-DELETE-MGMT-INFO      INCLUDED   1     003/000  05/11/77 14:17 UPD/DGC        SCOBOL       16  05/11/77
FMMD-ENVIRONMENT-DIVISION  INCLUDED   1     034/000  05/11/77 14:17 UPD/DGC        SCOBOL       20  05/10/77
FMMD-FILE-SECTION          INCLUDED   1     034/000  05/11/77 14:17 UPD/DGC        SCOBOL       15  05/10/77
FMMD-INTERPRET-DATA-VALUE  INCLUDED   1     034/000  05/11/77 14:17 UPD/DGC        SCOBOL       72  05/11/77
FMMD-INTERPRET-FMMD-VALUE  INCLUDED   1     032/000  05/11/77 14:17 UPD/DGC        SCOBOL       25  05/11/77
FMMD-NUMERIC-EDIT          INCLUDED   2     018/000  05/11/77 14:17 UPD/DGC        SCOBOL       14  05/11/77
FMMD-PROCEDURE-DIVISION    INCLUDED   1     041/000  05/11/77 14:17 UPD/DGC        SCOBOL       62  05/10/77
FMMD-READ-EDIT-SORT-TRANS  INCLUDED   1     014/000  05/11/77 14:17 UPD/DGC        SCOBOL       32  05/11/77
FMMD-SETUP-ACCOUNTING-INFO INCLUDED   1     005/000  05/11/77 14:18 UPD/DGC        SCOBOL       15  05/11/77
FMMD-STORE-EDIT-KEYWORDS   INCLUDED   1     038/000  05/11/77 14:18 UPD/DGC        SCOBOL       41  05/11/77
FMMD-WORKING-STORAGE-01    INCLUDED   1     054/000  05/11/77 14:18 UPD/DGC        SCOBOL      110  05/10/77
FMMD-WORKING-STORAGE-77    INCLUDED   1     043/000  05/11/77 14:19 UPD/DGC        SCOBOL       26  05/10/77
FMMVE                      MAIN             003/000  05/11/77 14:19 ORG/FHACD128   SCOBOL       18  04/20/77
FMMV-ENVIRONMENT-DIVISION  INCLUDED   1     004/000  05/11/77 14:19 UPD/DGC        SCOBOL       18  05/11/77
FMMV-FILE-SECTION          INCLUDED   1     005/000  05/11/77 14:19 UPD/DGC        SCOBOL       15  05/11/77
FMMV-MOVE-OLDPROJ-OLDLIB   INCLUDED   1     005/000  05/11/77 14:19 UPD/DGC        SCOBOL       50  05/11/77
FMMV-PROCEDURE-DIVISION    INCLUDED   1     005/000  05/11/77 14:19 UPD/DGC        SCOBOL       15  05/11/77
FMMV-WORKING-STORAGE-01    INCLUDED   1     006/000  05/11/77 14:19 UPD/DGC        SCOBOL      177  05/11/77
FMMV-WORKING-STORAGE-77    INCLUDED   1     003/000  05/11/77 14:19 UPD/DGC        SCOBOL       51  05/11/77
FMUPE                      MAIN             023/000  05/10/77 20:07 ORG/FHACD128   SCOBOL       18  04/20/77
FMUP-ADD-CHANGE-TEMP-DATA  INCLUDED   1     012/000  05/11/77 14:20 UPD/DGC        SCOBOL       34  05/11/77
FMUP-CHANGE-MGMT-UNIT      INCLUDED   1     027/000  05/11/77 14:20 UPD/DGC        SCOBOL       30  05/11/77
FMUP-ENVIRONMENT-DIVISION  INCLUDED   1     025/000  05/11/77 14:20 UPD/DGC        SCOBOL       18  05/10/77
FMUP-FILE-SECTION          INCLUDED   1     024/000  05/11/77 14:20 UPD/DGC        SCOBOL       15  05/10/77
FMUP-INITIALIZE-ADD-MGMT-UNIT INCLUDED 1    013/000  05/11/77 14:20 UPD/DGC        SCOBOL       15  05/11/77
FMUP-PROCEDURE-DIVISION    INCLUDED   1     024/000  05/11/77 14:20 UPD/DGC        SCOBOL       27  05/10/77
FMUP-WORKING-STORAGE-01    INCLUDED   1     033/000  05/11/77 14:20 UPD/DGC        SCOBOL      177  05/10/77
FMUP-WORKING-STORAGE-77    INCLUDED   1     031/000  05/11/77 14:20 UPD/DGC        SCOBOL       51  05/10/77
ITCLE                      CALLED           001/000  05/16/77 19:49 ORG/FHACD128   SCOBOL       39  04/20/77
ITCL-CHECK-COLLECTION-UNIT-KEY INCLUDED 1   001/000  05/16/77 19:49 UPD/DGC        SCOBOL       16  05/16/77
ITCL-INTERPRET-KEYWORD-VALUES INCLUDED 1    001/000  05/16/77 19:49 UPD/DGC        SCOBOL       53  05/16/77
ITCL-PROCEDURE-DIVISION    INCLUDED   1     001/000  05/16/77 19:50 UPD/DGC        SCOBOL       39  05/16/77
ITCL-STORE-EDIT-INPUT-KEYWORDS INCLUDED 1   001/000  05/16/77 19:50 UPD/DGC        SCOBOL       41  05/16/77
ITCL-WORKING-STORAGE-01    INCLUDED   1     001/000  05/16/77 19:50 UPD/DGC        SCOBOL       65  05/16/77
ITCL-WORKING-STORAGE-77    INCLUDED   1     001/000  05/16/77 19:50 UPD/DGC        SCOBOL       21  05/16/77
ITCL-WRITE-INPUT-KEYWORD-CARDS INCLUDED 1   001/000  05/16/77 19:50 UPD/DGC        SCOBOL       26  05/16/77
ITCL-WRITE-JCL-FOR-MDCOLLECT INCLUDED 1     001/000  05/16/77 19:51 UPD/DGC        SCOBOL       49  05/16/77
```

Figure 9. AUTHOR Report



```
DATE  06-13-77

                                       TIPTOP-PDL-HEADER



                      *    THE MODULE TIPTOP PROCESSES THE ADD A UNIT FUNCTION (ADUN)

                       TIPTOP
                           INITIALIZE PARAMETER-TABLE WITH SPACES
                           CALL OBUS
                           MOVE PGMR-NAME TO PARAMETER-TABLE
                           OPEN INPUT INPUT-CARDS.
                                 OUTPUT MESSAGE-FILE
                           MOVE "ADD" TO INPUT-FUNCTION
                           READ INPUT-CARDS
                           IF NOT END OF INPUT CARDS
                             CALL OBFN.
                             DO UNTIL NORMAL RETURN FROM OBFN
                               SEARCH PSL-FUNCTIONS
                                 WHEN PSL-FUNCTION IN THE TABLE EQUAL INPUT-FUNCTION
                                 SET FUNCTION-NUMBER TO INDEX OF TABLE.
                               INCLUDE PROCESS-FUNCTION.
                               CALL OBFN.
                             ENDDO.
                             CALL PRINT MESSAGE (END OF INPUT CARDS).
                             IF SPAWN JOB NEEDED
                               INCLUDE SPAWN-A-JOB
                             ENDIF.
                           ELSE
                             CALL PRINT MESSAGE (NO INPUT CARDS)
                           ENDIF.
                           CALL OLAF.
                           CLOSE INPUT-CARDS.
                                 MESSAGE-FILE.
                           STOP RUN.
```

Figure 10. DOCUMENT Report



```
PROJECT: EHAC0147              STRING -  TIPTOP                                                              PAGE   1
LIBRARY: PROVEN                                                                                              DATE: 06/11/77
SECTION: SOURCE                                                                                              TIME:    10:02

UNIT                           LINE COL         1         2         3         4         5         6         7         8
****************************** **** NO ----5----0----5----0----5----0----5----0----5----0----5----0----5----0----5----0
TIpTOp                            2 23      pROGRAM-ID.  TIpTOp.
                                  4 20         INCLUDE TIPTOP-ENVIRONMENT-DIVISION.
                                  8 20         INCLUDE TIPTOP-FILE-SECTION.
                                 10 20         INCLUDE TIPTOP-WORKING-STORAGE.
                                 14 20         INCLUDE TIPTOP-PROCEDURE-DIVISION.
TIPTOp-ENVIRONMENT-DIVISION    NONE
TIPTOp-FILE-SECTION            NONE
TIPTOp-PROCEDURE-DIVISION      STUB
TIPTOp-WORKING-STORAGE         STUB
```

Figure 11. Character Scan Output

3.2.6 Management Data Collection

The Management Data Collection and Reporting (MDCR) capability enables software projects utilizing the PSL facility to obtain statistical data reports on the status and progress of project activities. Figure 12 illustrates a typical MDCR Data Base composed of a single Management (MGMT) section library and multiple Source section libraries. The set of top units and included units present in these libraries constitutes the total code under development for a given project. Statistics relating to unit update activity are maintained in the individual Unit Accounting records. These "automatic" statistics may be collected and summarized through Management Data Functions.

A management plan is defined to describe the hierarchical organization of data reports whose generation is illustrated in Figure 13. Format units are established for each of the reported data levels described in the management data plan to determine the order and source of data items to be collected and summarized.

Module level data may be derived from essentially two data sources; that is, 1) unit accounting records and 2) manual inputs. Job level data is manually input only. Manual data can be added, changed, or deleted.

Manual inputs are also made to maintain exception check specifications. Any two numeric data items at the same format level may be subjected to a value "variance" check. The data item values are compared at the time the data is collected and a determination made as to whether an exception exists. If an exception does exist, pertinent information is added to the data collection unit so that management reports which are subsequently produced from that collection may flag the excepted data item.

A data collection is performed according to an indicated management data plan. The ensuing collection activity automatically verifies and cross-relates plan level elements, format requirements and data source availability. Data collection will procede until the entire management plan is processed and the collected data is stored in the designated MGMT section.

After data has been collected, a management report can be generated upon request. The management data report has a general format which provides appropriate identification of the level of data and the specific items for which values are printed. Figure 8 contains a sample management data report.

<table>
<tr><th>Applicable PSL Functions</th><th colspan="6">* * * * * PSL MGMT Section * * * * *</th></tr>
<tr><td></td><td colspan="6">Management Data Plan Unit (MDCR-PLAN)</td></tr>
<tr><td></td><td colspan="6">SYSTEM = BIGTOP
JOB = BIGJOB
SUBSYS=SUB1, MODULE=MOD1A, MODULE=MOD1B
SUBSYS=SUB2, MODULE=MOD2A, MODULE=MOD2B</td></tr>
<tr><td rowspan="2">MDFORMAT</td><td colspan="6">Management Data Format Units (MDCR-FORMAT-)</td></tr>
<tr><td>SYSTEM</td><td>SUBSYS</td><td>MODULE</td><td>JOB</td><td colspan="2">UNIT</td></tr>
<tr><td rowspan="2">MDUPDATE
MDXCHECK</td><td colspan="6">Management Data Input Units</td></tr>
<tr><td>BIGTOP</td><td>BIGJOB</td><td>SUB1</td><td>SUB2</td><td>MODIA</td><td>ETC.</td></tr>
<tr><td rowspan="2">MDCOLLECT
PURGE</td><td colspan="6">Management Data Collection Units</td></tr>
<tr><td colspan="3">MDCR-COLLECTION</td><td colspan="3">MDCR-ARCHIVE-001-(DATE1)
MDCR-ARCHIVE-002-(DATE2)
MDCR-ARCHIVE-003-(DATE3)</td></tr>
<tr><td colspan="7">* * * * * * PSL SOURCE Section(s) * * * * * *</td></tr>
<tr><td></td><td colspan="3">Top Units</td><td colspan="3">Included Units</td></tr>
<tr><td>ADD
CHANGE
MOVE
REPLACE
PURGE</td><td colspan="3">MOD 1A
MOD 1B
MOD 2A
MOD 2B</td><td colspan="3"></td></tr>
</table>

Figure 12. Management Data Base Structure

Management Plan Input vs. Management Reports Output

System = PSL-project
  Job = PSL-system-test
 Subsystem = PSL-main
  Module = BCTLE
  Module = PPLKE

.
.
.
.
.
.

 Subsystem = PSL-Function
  Module = ADUNE
  Module = CHUNE
  Module = CRFLE

.
.
.
.
.
.

 Subsystem = PSL-auxiliary
  Module = ADXEE
  Module = CHXXE

.
.
.

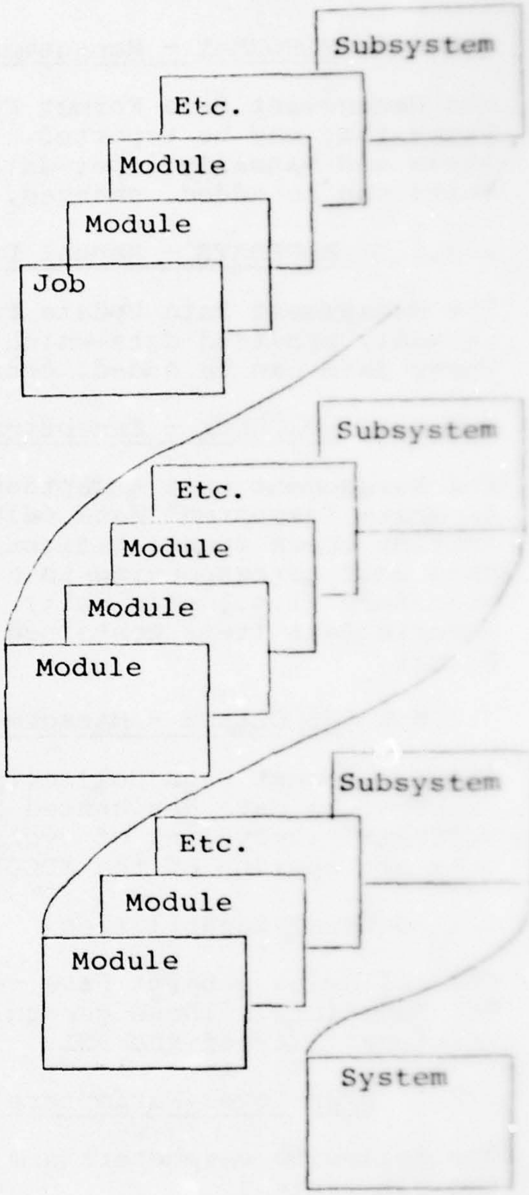


Figure 13. Management Report Structure

3.2.6.1 MDPLAN - Management Data Plan

The Management Data Plan Function is used to define the management data report structure.

3.2.6.2 MDFORMAT - Management Data Format Maintenance

The Management Data Format Function is used to define the data items that may be reported. Both automatically collected data items and manually input data items may be defined. Format units can be added, changed, or deleted by MDFORMAT as required.

3.2.6.3 MDUPDATE - Manual Data Maintenance

The Management Data Update Function is used to maintain the manually provided data which resides in management data units. These data can be added, changed, or deleted as required.

3.2.6.4 MDXCHECK - Exception Checking

The Management Data Exception Checking Function is used to annotate "excepted" data values in a management report. Exception check specifications are maintained in a management data unit corresponding to a denoted element in the report structure (i.e., plan unit). The specifications reference numeric data items contained in the associated record level format.

3.2.6.5 MDCOLLECT - Management Data Collection

The Management Data Collection Function is used to collect and archive the data designated by MDPLAN, MDFORMAT, MDUPDATE, and MDXCHECK. Recycling of cyclic data accumulations and archiving data are options of the MDCOLLECT Function.

3.3 General Capabilities

The following subject have general application over several PSL Functions. These general capabilities greatly enhance the serviceability of the PSL.

3.3.1 High-Level Parameters

The following parameters are cumulative during execution of the PSL system:

a.	PROJECT	Project name
b.	LIBRARY	Library name
c.	SECTION	Section name

d.	PASSWORD	Section password
e.	CLASS	Security classification code
f.	PGMR	Programmer name

Once the value for each of these keywords is established, either by a Function card or by default, it remains in effect. All of these parameters, except CLASS, may be replaced by another value by using the keyword on another Function card. A PARAM Function card is ordinarily used to establish one or more of these values, but the keywords may appear on any Function card for which they are valid. Use of a PARAM Function card eliminates the need for repetition of the same keywords on all subsequent Function cards utilizing those keywords.

Note on passwords:

A section password is required to access a section if the password was assigned when the section was created and if the processing will modifiy the contents of the section.

3.3.2 Multi-Library Search

The following Functions allow an optional multi-library search during retrieval of input:

a. COMPILE
b. LINK
c. EXECUTE
d. MDCOLLECT
e. MDPRINT

A maximum of nine libraries may be searched. Each particular library is identified by a unique project/library pair of keywords.

By permitting a multi-library search, the PSL provides the capability of automatically mixing development code with well-tested code for compilation and execution without altering the contents of the individual libraries. A multi-library search by the management data collection and reporting Functions permits data to be collected and reported from multi-projects as well as multi-libraries.

3.3.3 Independent Files

The PSL system is designed to store card-image data in blocks in a random file. This card-image data is read by special PSL access routines when it is retrieved by modules in the PSL system. However, if the data must sometimes be read by programs outside the PSL system, the data must be stored in a separate, or independent file. Examples of such data are unstructured code, which will be read by a standard compiler, and test data, which will be read by a user's program.

If a new unit is added to the SOURCE section for which the language is not a structured language supported by a precompiler in the PSL system, the system will create an independent sequential file for the data. Since an unstructured unit is not processed by a precompiler and INCLUDE statements are not resolved, the independent SOURCE unit must be a complete compilable set of code and may not be reduced to smaller units, as may be done with structured code.

Units in the OBJECT (which are output from a compiler or assembler) and LOAD (input to the Loader) sections are automatically created and maintained in independent files by the PSL system.

Whenever independent files are created, the user may provide File Management System (FMS) parameters to override the standard default FMS parameters provided by the PSL system or by FMS assumptions (See HIS File Management Supervisor File Directives).

3.3.4 Spawned Jobs

Certain PSL Functions, called job spawning Functions, require the execution of independent programs such as precompilers, GCOS compilers or loaders, or user written programs as part of their processing. To invoke an independent program, the PSL function retrieves a pre-stored JCL procedure, modifies the JCL and writes the modified JCL on a temporary file. Subsequent Functions may append additional JCL on this file. Also, JCL may be added directly to this file by the user with the JCL Function. When the end of the PSL input is reached and all the PSL Functions have been processed, a single separate job is spawned, using the temporary file of JCL as its single input stream.

The job spawning Functions are COMPILE, LINK, EXECUTE, MDCOLLECT, MDPRINT, and JCL. JCL procedures are provided with the PSL system for use with these Functions. The EXECUTE Function invokes a user's program using the JCL stored by the user in the JOB section of his library.

When a user's job invokes one of the job spawning functions, the $ IDENT card of the spawned job will be the same as that of the invoking job. The $ USERID card of the spawned job will contain:

1. The userid of the invoking job

2. The password which was used to initialize a PSL project whose name is the same as the userid

3. The security classification code found on the PARAM card or the system default value.

This means that for each userid which will run jobs invoking a job spawning Function, a PSL project must be initialized whose name is that userid.

3.3.5 PARAMCARD Keyword

A special keyword is provided for those PSL Functions which can result in a spawned activity. If the PSL system encounters a PARAMCARD keyword on one of these Function cards, a $ PARAM control card is generated in the JCL for the spawned activity. A 40-character literal field is picked up from the keyword-value and placed in columns 16 through 55 of the $ PARAM card.

The JCL procedures provided with the PSL system are designed to allow the user to specify certain options through the $ PARAM card such as destination of output, dump request in case of program abort, selection of output options for compilers, assemblers, and loaders.

3.2.6 Stub Generation

The PSL generates stubs for missing units under two conditions:

a. When structured source code is added to a library, a dummy SOURCE unit (SOURCE stub) is provided for any unit which is referenced in an INCLUDE statement, but has not yet been added to the library. (See description of the INCLUDE statement under subsection 3.3.8). Accounting information for the source stub is stored in the SOURCE section of the user's library.

b. When an object module is requested on an INCLUDE card in the user's loader control cards and the object module is not found by the PSL system, a dummy object module (OBJECT stub) is provided to the Loader by the system. A message will be printed on the system output file when the object stub is executed. The object stub is not stored in the user's library.

The object stub enables program execution with partially completed code without requiring the individual programmer to create special code which will subsequently be replaced.

3.3.7 Error Handling

When an error is encountered on a Function card, an error message is generated and the remaining keywords for that Function are scanned, but no library processing is done. Input is read and printed, but not processed, until the next Function card is found. If an error is found on a PARAM Function, no processing takes place until another PARAM Function card is found.

If an error occurs while processing a Function, the PSL system will attempt to undo whatever processing has taken place and will then terminate the Function. Advisory and error messages (see Reference B) will be generated. Processing will resume at the next Function card.

3.3.8 INCLUDE Statements

An INCLUDE statement is used in source code and with loader control cards to refer to a unit which will be retrieved and substituted in-line for the INCLUDE statement.

The INCLUDE statement may be used in the SOURCE, PDL, and LINK sections as follows:

a. SOURCE - When the INCLUDE statement is used as a line of code in a SOURCE unit, it refers to a lower-level unit of SOURCE code which will be substituted in-line for the INCLUDE statement. Such INCLUDE statements are resolved by a preprocessor before the code is passed to the compiler. Therefore, INCLUDE statements may only be used in modules which are written in a structured language, such as SCOBOL or SPFORT, which are supported with preprocessors in the PSL system. Independent units may not be INCLUDED by other units and should not contain INCLUDE statements. When a non-independent unit is added to a SOURCE section, or modified, the INCLUDE references are checked. If an included unit does not exist in the section, a SOURCE-stub unit is created and is tagged with the name and language of the including-unit. When the including-unit is compiled, the lower-level unit is a stub, an appropriate statement is inserted by the preprocessor to generate an output message when the code is executed. INCLUDE statements may be nested to a depth of 50 levels.

b. PDL - The INCLUDE statement is processed in the PDL section in the same way as in the SOURCE section. A Program Structure report may be generated for a unit in the PDL section.

c. OBJECT - When the INCLUDE statement is used with Loader control cards in the LINK section, it refers to a compiled unit (a module) in the OBJECT section which will be inserted into the input stream passed to the Loader. If the unit is not found in the libraries selected on the LINK (or EXECUTE) Function card, an OBJECT-stub unit is substituted in its place (with the unit-name as the external-name of the stub). This unit appears as a "STUB" in the load map of the user's program and a message will be printed when the OBJECT stub is executed.

3.3.9 Management Data Cycling

Data cycling is the periodic resetting of those management data items for which values have been accumulated over a period of time. Data cycling can be optionally defined to permit the user to specify the period for which values are to be accumulated before being reset.

"Cyclic" management data is first implemented in the Unit Accounting Record for PSL sections in which the MGMT=YES option is indicated. Two cyclic data items are maintained in that record: lines input per cycle and number of updates per cycle. The determination as to when the specified cycle period has elapsed is made when a management data collection is performed. If the number of days specified for the cycle period is equal to or exceeds the number of days elapsed since the accumulation of cycle statistics was begun, the cyclic data item contents will be reset to zero (i.e., recycled) after the collection of that data. A new data cycle is started and cycle duration is determined from that date for all SOURCE section units linked through the management plan; that is, through the module names listed in the management plan unit.

3.3.10 Management Data Archiving

Archiving enables the user to retain previously collected data for a historical review of project status. The determination to automatically archive is made when the data is collected, based upon the current date, the start-of-cycle date, and the cycle duration. Actual archiving is not performed, however, until the next data collection is performed so that the most recent data collection remains the "current" data collection until such time as it is archived and/or replaced.

Each archived unit is uniquely identified at the time of archiving by affixing a suffix to its basic name. This suffix consists of a three digit serial number and a six character date. The assigned serial number (starting with 001) is increased by one for each subsequent collection archived. The six character date, given in MMDDYY format, signifies the day on which the data collection was performed.

Archives may be retained on disk or written to a backup tape. Tape-stored archives can be restored from tape for printing at some later time.

3.3.11 Transportability

The design and implementation of PSL permits transportability between WWMCCS sites. To install and maintain PSL, ANSI COBOL and ANSI FORTRAN compilers are required as part of the site inventory. If FORTRAN is not used at a site, it is possible to exclude the FORTRAN precompiler without impacting the remainder of PSL operation.

Only the PSL load modules are required to be disk-resident. After installation, both source and object modules may be saved on tape thus not impacting disk storage at a site. The size of the users' libraries are not a function of PSL, but rather are limited by the available storage at any particular site.

Phase I PSL required 44,000 words of core for execution. By judicious restructuring of the overlays, by removing all "debug" print, and by other reduction techniques, Phase II, although containing more capabilities than Phase I, can be executed in 37,000 words of core. The single exception is when processing Structured FORTRAN (SPFORT). Since the GFE FORTRAN Precompiler is written in FORTRAN while the rest of PSL is written in COBOL, significantly more core, (i.e., 43,000 words) is required.

Since the interfaces between PSL and the GCOS Operating System are simple and straightforward, PSL operation is reasonably immune to GCOS changes. The only problems to date have been associated with changes in userid, password, sign-on procedures and security classification. The use of ANSII COBOL and the simple Operating System interface have provided unexpected advantages in transportability between types of computers. This relative transportability is expected to be demonstrated in a pending contract under which the PSL will be converted to operate on the UNIVAC 1100 series for the Defense Mapping Agency.

Section 4

PSL SYSTEM TESTS

Both Phase I and Phase II software deliveries were prefaced by a system test which demonstrated the PSL capabilities being delivered. A Test and Implementation Plan (Reference C and D) was delivered 90 days prior to each test.

Appropriate test data were developed and stored in PSL 'EST sections. Listings of these data were included as an appendix to each Test Plan. Updated listings were available at the system test so that participants and observers could easily follow the test sequence.

4.1 Capabilities Demonstration

A capabilities demonstration was given at Andrews Air Force Base on 24 June 1976. Background material on structured programming and programming support libraries was presented. The PSL system design was discussed and each Phase I Function was presented including input-card-format and available options. For all Functions completed at that time, computer output was available for perusal. Listings of all completed modules were displayed. This demonstration marked the first interface between IBM and the CCTC distributing agent. The completeness of the PSL Functions at that point in development prompted subsequent decisions to release Phase I as a useable tool.

4.2 Phase I System Test

The Phase I System Test was conducted on 9 September 1976 at Andrews Air Force Base. The first item in the agenda was a briefing on PSL installation. (Since the installation is a lengthy process, it was not part of the system test).

The Phase I System Test was conducted according to the Phase I Test and Implementation Plan. Although PSL is a batch system, development occurred via terminal interfacing with the GCOS time-sharing capability. Since all card decks were internally stored, the system test was also conducted via terminal. However, the terminal was located in a conference room at Andrews Air Force Base. A test was submitted and results were retrieved from the computer operations facility. When the results had been analyzed to the point of high confidence, the next test was entered.

Eight tests were scheduled to be conducted. However, Test 3 was omitted because four PSL Functions were not completed at that time - BACKUP, RESTORE, MOVE and REPLACE. With those exceptions, the Phase I System Test was successfully completed and the software package subsequently delivered.

The results of the Phase I System Test are presently in Reference E, Phase I Test Analysis Report.

4.3 Phase II System Test

The Phase II System Test was scheduled for June 2 and 3, 1977. The highlight of the test was a briefing and subsequent demonstration of the Management Data Collection and Reporting Subsystem. As in Phase I, the Phase II System Test was conducted at Andrews Air Force Base via terminal from a conference room. Printer output was given high priority and retrieved for immediate perusal.

The Phase II System Test was conducted according to the Phase II Test and Implementation Plan. The Phase II System Test included the Phase I capabilities demonstrated in September 1976, the four Phase I Functions which were not demonstrated at that time, and the Phase II capabilities.

Due to the associated briefings and the great interest shown by the observers, the system test was not completed in two allotted days. A third day was required to complete all tests. There were no exceptions at the final completion of the system test. The software package was subsequently delivered.

Results of the Phase II System Test are presented in Reference F, Phase II Test Analysis Report.

Section 5

PSL DELIVERIES

The following items were delivered for each phase:

5.1 Phase I Deliverables

The following items were delivered during Phase I development:

a. Test and Implementation Plan, CDRL Item A006, June 1976

b. Users Manual, CDRL Item A003, July 1976

c. Installation Manual, CDRL Item A00B, July 1976

d. Software, Phase I, CDRL Item A00A, September 1976

e. Test Analysis Report, CDRL Item A007, November 1976

5.2 Phase II Deliverables

The following items were delivered during Phase II development:

a. Test and Implementation Plan, CDRL Item A006, March 1977

b. Users Manual, CDRL Item A003, April 1977

c. Installation Manual, CDRL Item A00B, March 1977

d. Software, CDRL Item A00A, June 1977

e. Test Analysis Report, CDRL Item A007, August 1977

5.3 Other PSL Deliverables

Two PSL documents were delivered once during the execution of the contract:

a. Program Maintenance Manual, CDRL Item A005, August 1977

b. Technical Report, CDRL Item A008, August 1977

A monthly status report was delivered, CDRL Item A001. Two related documents were delivered ten months after the start of the contract:

a. COBOL Precompiler Manual, CDRL Item A004, August 1976

b. Program Production Library (PPL) Operations Manual, CDRL Item A004, July 1976

These documents were not part of the original contract, but were added by Supplemental Agreement P00001.

5.4 Contract Amendments

The contract began September 3, 1975 with the schedule shown in Figure 14. In May 1976, Amendment P00001 made the following contractual changes:

a. The Phase I schedule was slipped for three months such that Phase I software delivery was scheduled for 3 September 1976.

b. The standalone COBOL Precompiler Installation Manual and software were added to the CDRL.

c. The Program Production Library (PPL) Operations Manual was added to the CDRL.

d. Error data collection was initiated.

Since the Phase I PSL was a useful and operational program which provided basic library maintenance, a set of decisions about PSL implementation were made during the summer of 1976. Phase I PSL was to be distributed to the five WWMCCS sites which were participating in the WWMCCS Structured Programming Experiment. Distribution is under the jurisdiction of the WWMCCS ADP Directorite, Command and Control Technical Center (CCTC), Code 411. To prepare for PSL installation at the five sites, a series of pre-installation briefings were scheduled. Additionally, the necessity of providing support to PSL users was recognized. The five sites involved in the Structured Programming Series are:

a. SAC Headquarters, Offutt Air Force Base, Nebraska

b. MAC Headquarters, Scott Air Force Base, Illinois

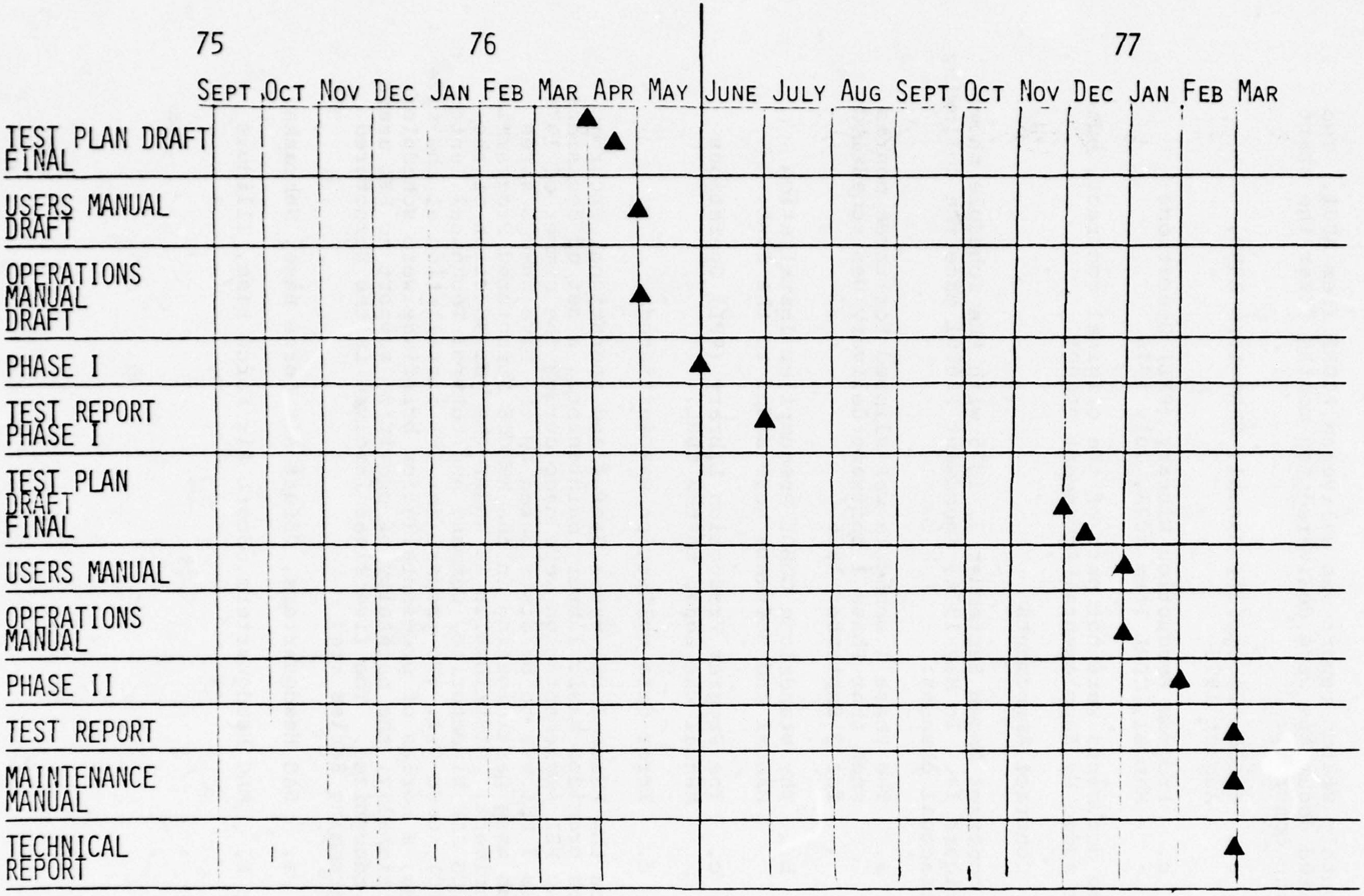


Figure 14. Original PSL Schedule of Deliverables

c. Army Command Center, Ft. Gillam, Georgia

d. CCTC, Pentagon, Washington, D.C.

e. NARDAC, Navy Yard, Washington, D.C.

Unofficial Phase I users were ADCOM, Colorado Springs, Colorado and AFSC, Andrews Air Force Base, Maryland.

To provide the necessary support, two new amendments were added to the contract:

a. To incorporate the GFE FORTRAN precompiler STRUCTRAN 1) into the Phase II PSL and provide the necessary support routines, and 2) to provide pre-installation briefings at the five official sites and at ADCOM.

b. To provide user support of Phase I PSL in the areas of answering questions, accepting and resolving problem reports, and assisting users in implementing their own unique programming projects.

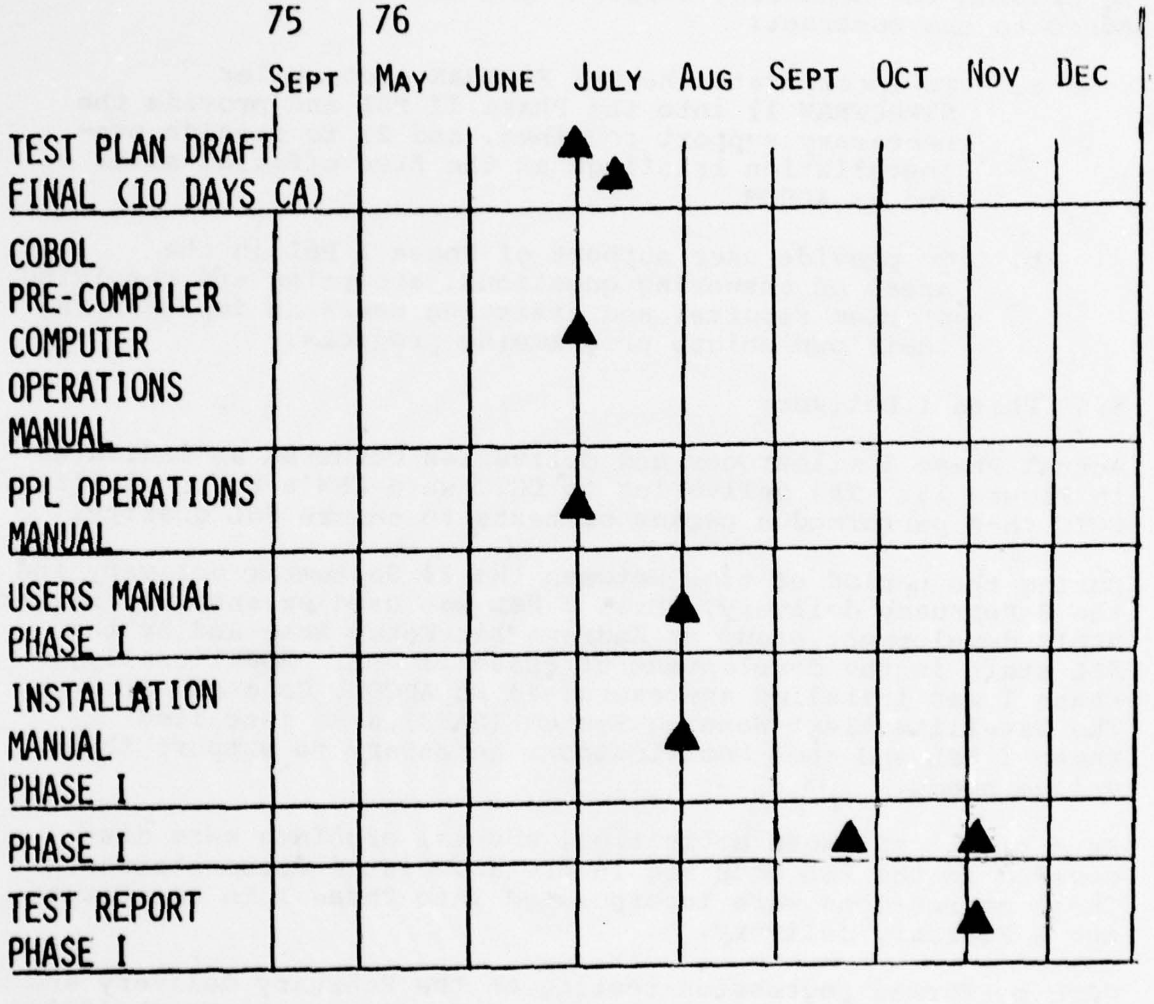
5.5 Phase I Delivery

Actual Phase I milestones and deliveries occurred as indicated in Figure 15. The deliveries to CCTC were IBM's responsibility. CCTC then performed a series of tests to ensure PSL quality.

During the period of time between the 24 September delivery and the 8 February delivery, Phase I PSL was used extensively by the MASIS development group at Andrews Air Force Base and by the PSL staff in the development of Phase II PSL. Additionally, Phase I was installed and exercised at ADCOM, Colorado Springs. The Satellite Alert Warning System (SAWS) also installed Phase I PSL and made modifications necessary to support their unique needs.

As a result of these activities, several problems were discovered in the PSL code and in the associated documentation. These corrections were incorporated into Phase I to constitute the 8 February delivery.

CCTC performed regression testing on the February delivery and released Phase I PSL to the designated sites on 21 March 1977.



	75	76							
	SEPT	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
TEST PLAN DRAFT FINAL (10 DAYS CA)				▲ ▲					
COBOL PRE-COMPILER COMPUTER OPERATIONS MANUAL				▲					
PPL OPERATIONS MANUAL				▲					
USERS MANUAL PHASE I					▲				
INSTALLATION MANUAL PHASE I					▲				
PHASE I							▲	▲	
TEST REPORT PHASE I								▲	

Figure 15. Actual Phase I Deliverables

5.6 Phase II Delivery

Due to the extended Phase I schedule and to the volume of work to be performed for Phase II, a two-month schedule adjustment was requested. The adjusted schedule place the Phase II software delivery on 3 May 1977 instead of 3 March. It was expected that a May delivery would also be able to incorporate 1) a JOVIAL (JOCIT) precompiler which was under development, and 2) additional corrections which might result from more extended use of Phase I by the designated WWMCCS sites.

The Phase II development plan, however, was delayed in almost all areas:

a. Computer problems in February and March, 1977 in conjunction with a new system software release caused significant delays at a critical point in Phase II development.

b. The JOVIAL precompiler was not completed in time for Phase II.

c. Distribution of Phase I to the designated WWMCCS site was delayed so that no feedback was received from those sites.

d. The extent of Phase II required additional time, thus causing further schedule slips. The Phase II System Test was successfully conducted on June 2, 3 and 14, 1977.

The actual Phase II deliveries were made according to Figure 16.

5.7 Post Phase II

Two further amendments have been made to the original contract for Post Phase II delivery. These amendments are mentioned here only for information purposes.

a. Incorporation of the GFE JOVIAL (JOCIT) precompiler with associated printing support. Also included in this amendment is the preparation of appendices to the Users Manual, Installation Manual, and Program Maintenance Manual describing the PSL installation at RADC under the commercial GCOS environment.

MONTH DELIVERABLES	75 Sept.	77 Jan.	Feb.	Mar.	Apr.	May	June	July	Aug.	Sept.
TEST PLAN * DRAFT				▲						
USERS MANUAL *					▲					
INSTALLATION MANUAL *				▲						
PHASE II							▲			
TEST REPORT									▲	
MAINTENANCE MANUAL									▲	
TECHNICAL REPORT									▲	

* Manual Updates were provided as part of the Phase II delivery.

Figure 16. Actual Phase II Deliveries

b. User Support for both Phase I and Phase II PSL from 1 August 1977 until 1 March 1978. The sites to be supported under this amendment are:

(1) SAC Headquarters, Offutt Air Force Base, Nebraska

(2) MAC Headquarters, Scott Air Force Base, Illinois

(3) Army Support Command, Ft. Gillam, Georgia

(4) CCTC, Pentagon, Washington, D.C.

(5) NARDAC, Navy Yard, Washington, D.C.

(6) RADC, Griffiss Air Force Base, New York

(7) AFSC, Andrews Air Force Base, Maryland

(8) ADCOM, Colorado Springs, Colorado

Section 6

PSL INSTALLATION

The initial installation of PSL at a site is performed from a save tape generated by the distributing agency. For WWMCCS stes, CCTC is the distributing agency; for non-WWMCCS sites, IBM is the distributing agency with RADC approval. Installation procedures and associated information are provided in Reference G, PSL Installation Manual.

Since PSL is written in Structured COBOL, a COBOL precompiler is necessary to recompile those modules for which site changes are required (see Reference G). The restore of the Installation Tape provides a useable PSL, including the COBOL precompiler. Changes are than made to the following PSL program modules.

The PSL itself is used to compile these modules and generate new load modules. When the installation procedures are completed, the PSL source code, and object code if desired, can be saved on tape thus freeing disk space for other uses.

An installation test is provided as part of the installation package. The installation test is a subset of the system test. Correct execution of the installation test places a high degree of confidence in a successful installation.

Pre-installation briefings were presented by IBM at all indicated WWMCCS sites. The Installation Manual (Reference G) attempted to simplify the procedures such that site personnel could install the PSL with minimal telephone support from IBM. A Users Manual (Reference B) is available to assist PSL users. IBM, under the User Support amendments, provides letter, telephone, or in-person support as needed.

The PSL has also been installed under the commercial GCOS environment, both for the SAWS project (an unofficial user) and at RADC. Differences in the COBOL compiler between WWMCCS and the commercial version necessitated recompiling all PSL modules. A linking problem arose under the commercial environment that did not exist in the WWMCCS environment.

Upon resolution of these discrepancies, PSL operated successfully in the commercial environment. SAWS development utilizes a modified PSL, modified by SAWS personnel. RADC assumes no responsibility for the SAWS version of PSL.

Section 7

PSL STATISTICS

The statistics for Phase I PSL are impressive and indicate a productivity rate of 20 lines of code per day.

Lines of source code, Phase I - 33000

Number of documentation pages, Phase I - 360

Number of errors in Phase I since system test - 74

Phase II statistics do not yet include any errors, since Phase II use has been minimal.

Lines of source code, Phase II - 15000

Number of documentation pages, Phase II - 700

Appendix A contains a summary of PSL problem reports to date.

Appendix A. PSL PROBLEM REPORTS

The following is a complete list of PSL Problem Reports (PRs) which were submitted against the Phase I PSL delivery. An analysis of each problem has been made and the necessary corrections have been incorporated in the Phase II PSL delivery.

PR#	Date Opened (Location)	Problem Description
1	6/7/76 Rosslyn	Extra character is printed at the beginning of a line when compressed data is read.
2	8/20/76 Rosslyn	Unit key in accounting record of stub is zero.
3	8/20/76 Rosslyn	Include processing invoked for INDEP unit.
4	8/20/76 Rosslyn	Source list indents % INCLUDE as continuation instead of new statement.
5	8/20/76 Rosslyn	Cannot get file classification when setting up to create index file.
6	8/20/76 Rosslyn	There is no check for structured programming errors in the Program Structure Report.
7	8/24/76 Rosslyn	Cannot print index of last OBJECT section created. No error message printed.
8	8/24/76 Rosslyn	Compile function cannot assign section file for object section.
9	8/24/76 Rosslyn	Data on INDEX print of accounting information for object units contain all zeros.
10	8/27/76 Rosslyn	Precompiler puts an extra line at end of generated code.
11	8/26/76 Rosslyn	Abort attempting to modify independent file.
12	8/27/76 Rosslyn	Error messages from loader "UNDEFINED SYMBOL PCLUE, PGFLE" when modules PGUN, DLUN are put in system.

PR#	Date Opened (Location)	Problem Description
13	9/13/76 Rosslyn	Create function message says "MISSING KEYWORD SECTION" when PROJECT is missing.
14	9/10/76 Rosslyn	Precompiler does not print "PROJECT LIBRARY" at start of module.
15	9/13/76 Rosslyn	Footnote on program structure map has incorrect number for "more than one statement on a line".
16	9/13/76 Rosslyn	Message 77 has PROJECT/LIBRARY/ SECTION line but no data filled in.
17	8/30/76 Rosslyn	Section code not passed to PRPS. Message 205, 6, 7 are printed in PS activity with no section code.
18	9/14/76 Rosslyn	After CHANGE, language of unit is 000000.
19	11/1/76 ADCOM	Abort when ** EXECUTE directive encountered.
20	11/1/76 Rosslyn	Msg "INVALID" password from **COMPILE when correct password is provided on ##PARAM card.
21	11/10/76 Rosslyn	Generated JCL from **EXECUTE OBJECT option, put lud "00".
22	11/22/76 ADCOM	COMPILE flag in wrong column of $ COBOL card in skeleton JCL-SCOBOL. Also caused punched object deck output when OBJECT=NO.
23	10/25/76 SAWS	A parameter to a called function was not level 77 or 01.
24	11/11/76 Rosslyn	When unit other than stub-unit contains no data lines the header line for unit is not printed.
25	11/11/76 Andrews	Syntax error is printing before card it is referencing.
26	12/7/76 Andrews	Structured COBOL identation program, indents lines in "REMARKS" paragraph.

PR#	Date Opened (Location)	Problem Description
27	12/7/76 Andrews	When "INCLUDE" statement for stub unit is last statement of a paragraph, compiler error results because generated "DISPLAY" statement.
28	12/7/76 Rosslyn	SOURCE function does not check for illegal sections LOAD or OBJECT. May abort if user specifies LOAD or OBJECT.
29	12/7/76 Rosslyn	Use of dummy **PARAM card when $USERID is not same as PROJECT. System could assume user password should come from a PROJECT whose name is equal to USERID.
30	12/8/76 Rosslyn	Names of section which had previously been purged by FMS, or due to an error condition developing later in PGSC remained in the project index.
31	12/10/76 Rosslyn	When deleting a stub, PCLU uses HIGHER-UNIT-BLOCK-NBR to release data block instead of FIRST-DATA BLOCK-PTR.
32	12/10/76 Andrews	Abort in **ADD when space exhausted on INDEP unit causes existence of unit which can neither be ADDed nor PURGEd.
33	12/10/76 Andrews	Unit is purged but higher-unit name is not removed from higher-unit-list in included unit.
34	12/10/76 Andrews	When SPLENGTH defaults to 50, structured listing flags excess lines over 210.
35	12/10/76 Andrews	When INCLUDE statements are deleted PCLU is not being called when keyword "SECT=" is not specified on change function card.
36	12/10/76 Andrews	When FMS file size keyword equal "unlimited" maximum file size does not get set properly.

PR#	Date Opened (Location)	Problem Description
37	12/10/76 Andrews	**BACKUP of section fails when INDEP unit encountered which was added by a USERID different from PROJECT-NAME.
38	12/10/76 Andrews	Source listing shows an included unit with "INCLUDED IN:....b̸".
39	12/14/76 Andrews	Error "PSL073 nesting of READS exceeds 5 levels" occurs in precompile and program structure when "INCLUDE"s nested 6 deep.
40	12/14/76 Rosslyn	Insufficient space in random section file for **CHANGE cause PS block #0 to be overwritten, destroying CONTROL-BLOCK-info.
41	12/17/76 Rosslyn	Precompiler ABORTS when last statement of module is DO WHILE.
42	12/17/76 Andrews	ERR059 printed by **CHANGE with multiple deletes.
43	12/17/76 Andrews	2 runs - **TERMINATE, *CREATE on same section produces section such that index-entry exists in project index but section random file does not exist.
44	1/5/77 SAWS	Change processing not sorting records before processing.
45	10/14/76 Rosslyn	Independent unit name remains in SECTION index if unit has been purged outside of PSL. Cannot remove name or readd unit.
46	1/12/77	Change function updates OBJECT section when it is specified.
47	1/14/77 Andrews	When unit is moved, msg. ERR55 printed but old unit remains in library.
48	1/17/77 RADC	Installation manual should include tape, label, density, track information.
49	1/17/77 RADC	Installation manual should specify to compile program PPLK page 6,8.

PR#	Date Opened (Location)	Problem Description
50	1/17/77 RADC	User's manual pages 60, 65, 71, 75 state *PROJECT must equal $USERID* incorrectly. Page 75 says UNIT keyword is not required-incorrect.
51	1/18/77 Andrews	When included unit name is 30 characters long an error is generated and DLUN processing ends without further processing of unit.
52	12/2/76 CCTC	Strange error message and indications that source and object modules did not contain the same code.
53	12/2/76 CCTC	PSL Test files were not purged from the installation tape.
54	1/21/77 Rosslyn	SP-error flag not printed on program structure report.
55	1/24/77 CCTC	General read permissions not assigned to catalog FHACD146. Makes system inaccessible to other users.
56	1/24/77 CCTC	Cannot always override $SYSOUT cards in spawned JCL to put in ORG for return of sysout to remote terminal.
57	1/24/77 CCTC	FORTRAN code entered in library is printed by FORTRAN compiler with 0000 in columns 75-78.
58	1/24/77 CCTC	FORTRAN procedure uses compiler FORTRAN. CCTC testers feel that FORTY is more versitile compiler.
59	1/24/77 CCTC	Compile procedures have two replaceable parameters on $COMPILE cards. CCTC would like 6.
60	1/24/77 CCTC	Job spawning functions **EXECUTE, **COMPILE, **LINK, require JCL keywords in columns 72-80 to be blank filled. Would be nice if numeric fill were also accepted.

	Date Opened	
PR#	(Location)	Problem Description
61	1/24/77 CCTC	Operation receives message "SYSOUT LINES EXHAUSTED" even though SYSOUT LIMIT has not been exceeded.
62	1/24/77 Rosslyn	Following files should not be part of FHACD146: /AAA; /BBB; /FFF; /BACKUP; /PSLPRG/SOURCE/INSTTEST.
63	1/31/77 CCTC	Installation test data produces compile error when test is run.
64	2/18/77 Andrews & Rosslyn	COBOL paragraph name immediately following IF statement is not recognized as end of condition and produces incorrect generated code.
65	2/18/77 Rosslyn	Incorrect indentation of COBOL code results when paragraph name or other code before column 12 occurs within range of a structure figure.
66	2/24/77 Rosslyn	When PSL job is submitted through time-sharing, the execution report and system output (P*) cannot be sent directly to the central site printer.
67	3/14/77 CCTC	When indentation caused code to be right adjusted past end of page, excess code is truncated.
68	3/15/77 Rosslyn	RDPS and RDCM stop searching multiple libraries when a stub is found. When a unit to be modified is moved to a new library, its included units are probably stubs, not real units.
69	3/29/77 CCTC	Line 845 'EQUAL TO' should 'GREATER THAN' Line 1349 'ERROR-MESSAGE-NUMBER' should be 'ERROR NUMBER'.
70	3/29/77 CCTC	After line 172 in BCTL there must be a new line. '05 FILLER PIC X (28) VALUE SPACE'; otherwise problems if ESPA even returns a maximum length value for print classification.

PR#	Date Opened (Location)	Problem Description
71	3/29/77 CCTC	ADUN, ADXE, BKLB, CHXE, DLUN, MVUN, PGSC, PGUN, RPUN, RSLB all have 'program - ID' paragraph header beginning in area B rather than area A - syntactically invalid. ADXE and CHXE have 'SOURCE-COMPUTER', 'OBJECT-COMPUTER', and 'SPECIAL NAMES' headers in area B rather than area A.
72	3/29/77 CCTC	Line 66, 'VALUE 99' should be changed to 'VALUE 99 THRU 99999999' to prevent odd listings at high level of indentation.
73	3/29/77 CCTC	Line 1270, the FILLER should be 'PIC x (20)' to have a syntactically valid program and to avoid strange listings at high levels of indentation.
74	4/10/77 RADC	GFRC abort in PPLK reading object unit.

APPENDIX B. SUPPLEMENTAL TECHNICAL REPORT

New and enhanced PSL functional capabilities were developed and implemented under RADC contract F30602-77-C-0249 subsequent to the Phase II delivery of the PSL system. The following is a technical report on the work performed and deliveries made in the Post Phase II time period.

a. Project References

1. Programming Support Library (PSL) Users Manual, February 1978.

2. Programming Support Library (PSL) Maintenance Manual, February 1978.

3. Programming Support Library (PSL) Installation Manual, February 1978.

4. Programming Support Library (PSL) Test and Implementation Plan, February 1978.

5. Programming Support Library (PSL) Test Analysis Report, February 1978.

b. PSL Capabilities

1. CREATE - Create a Section

Within a library, the PSL system allows the user to create the following addition in the PSL section:

- USER - This section contains and accounts for data that may be input to and output from standard PSL and/or special user operations. Data accessible to standard PSL operations is accounted for via the section index; data created and maintained by user programs only is accounted for via the section directory.

2. CHANGE - Change a Unit

An enhancement was developed and implemented to convert a random-stored PSL unit to an independent (sequentially-stored) unit when and if section space is exhausted during a unit update. The following advisory message is printed when such occurs:

ADV086 SECTION SPACE EXHAUSTED. UNIT TYPE CHANGED TO INDEPENDENT.

In this case, the user may increase the space allocated to the specific section and restore the converted unit to its original random-stored unit type by performing the following:

(a) use the BACKUP Function to save the section at which time the converted unit is written to the backup file as if it were its original unit type.

(b) use the TERMINATE Function to release the existing section.

(c) use the CREATE Function to create a larger section space.

(d) use the RESTORE Function to restore the contents of the section from the backup file written in the previous operation.

3. PERFORM - Perform Job Control Cards

The PERFORM Function is primarily used to interface data between PSL and non-PSL operations. Job control cards contained in a user-designated JOB section unit are output to the spawned job file. These job control (e.g., $FILE) cards in which the user has specified a flagword (starting in column 73), are processed to create independent unit files and update related PSL section accounting information as well as output $PRMFL cards with the necessary FMS catalog/file information to control input/output operations that access the independent files.

4. PRECOMPILE - Precompile a Module

The PRECOMPILE Function is used to read the top and INCLUDEd units of a source code module and produce an output containing the total module source code. This output may subsequently be directed to PSL storage or a user-designated program procedure.

5. COMPILE - Compile a Module

The COMPILE Function is enhanced to accommodate the compilation of JOVIAL language source code. Added keyword inputs are accepted to provide for the storage of COMPOOL symbolic tables and their reference in main program compilations. Structured JOVIAL code is preprocessed to produce code acceptable to the JOVIAL (i.e., JOCIT) compiler. The COMPILE Function is additionally enhanced to preprocess non-structured COBOL, FORTRAN, JOVIAL and GMAP source modules from PSL random-block storage to collect included code for input to the compilation activity.

6. SOURCE - Print a Card-Image Unit

The SOURCE Function is enhanced to print structured JOVIAL source code in an indented format and to print structured figure error messages when errors are detected in the structured JOVIAL unit code.

7. PURGE - Purge a Unit

The PURGE Function has been expanded to delete files listed in the section directory (primarily of the USER section). The keyword FILE is used to denote the keyword value naming the file to be purged.

c. Supplemental System Test

A supplemental (Post Phase II) system test was scheduled for 24 February 1978. As in Phase II, the Supplemental System Test was conducted at Andrews Air Force Base via terminal. Printer output was retrieved from computer operations for immediate perusal. There were no exceptions at the final completion of the system test. Results of the Supplemental System Test are presented in an Appendix to the Phase II Test Analysis Report.

d. Post Phase II PSL Deliveries

The following items were delivered:

1. Users Manual, CDRL ITEM A003, February 1978.
2. Program Maintenance Manual, CDRL ITEM A005, February 1978.
3. Installation Manual, CDRL ITEM A004, February 1978.

4. Test and Implementation Plan, CDRL ITEM A006, February 1978.

5. Test Analysis Report, CDRL ITEM A007, February 1978.

6. Software, CDRL ITEM 0001, February 1978.

7. Technical Report, CDRL ITEM A008, February 1978.

e. PSL Statistics

Five new modules were incorporated into the PSL system containing 5185 lines of code as was determined using the Management Data Collection and Reporting capability and approximately ten existing modules were modified to include enhancements related to newly developed capabilities and the resolution of reported system deficiencies.

f. PSL Problem Reports

All of the following reported problems have been resolved and the necessary corrections are incorporated in the Post Phase II PSL delivery.

PR#	Date Opened (Location)	Problem Description
1	8/22/77 (Andrews)	The MODIFY Subfunction to the CHANGE Function does not treat the absence of source data input as an error.
2	8/31/77 (Andrews)	The INSERT Subfunction to the CHANGE Function makes an improper insertion when the line number referenced is also subject to modification.
3	9/15/77 (Andrews)	When the CREATE Function is used to create a random file, the specification of unequal minimum and maximum file size parameters sometimes causes a problem.
4	10/1/77 (Rosslyn)	When structured COBOL code extends beyond print column 99, it is truncated and not printed.
5	10/1/77 (Rosslyn)	A program abort occurs while printing a structured COBOL unit when structured verbs start prior to column 12.

	Date Opened	
PR#	(Location)	Problem Description
6	10/12/77 (Rosslyn)	Erroneous INCLUDE statements submitted to the ADD Function cause extraneous characters to appear in the flagged INCLUDE statement line.
7	12/15/77 (Rosslyn)	The use of the $PARAM card to submit replaceable parameters on a $JOVIAL card results in a JOCIT compiler error.
8	12/20/77 (ADCOM)	When a PSL section file has insufficient space to complete an update, the updated PSL unit is truncated.
9	1/15/78 (Rosslyn)	When no system name is specified in the Management Data Plan, a line of zeroes appears in the management data report where the system name would normally be printed.
10	1/15/78 (Rosslyn)	When no system name is specified in the Management Data Plan, subsystem statistics are not collected.
11	1/31/78 (Andrews)	When the SECT=ALL option is used with the RESTORE Function, an inappropriate error message is generated.
12	2/20/78 (RADC)	See pages B-6 and B-7.

PSL System Installation at the RADC GCOS Facility

There are two identified differences between the installation requirements at a WWMCCS facility and the RADC GCOS facility:

1. The object modules produced from COBOL source code compiled at a WWMCCS facility cannot be executed at a GCOS facility because dynamic calls are issued for COBOL subroutines that are not present in the GCOS system library. All COBOL source code modules must therefore be compiled at the GCOS facility to produce object code that is compatible with the GCOS system library.

2. The COBOL sort/merge routine entry point name is CSRT2 at a WWMCCS facility but is CSORT at the RADC GCOS facility.

Installation of the post Phase II PSL system was attempted at the RADC GCOS facility as of February 1978. Recompilation of the COBOL source code modules was accomplished and the entry point name CSORT was substituted for CSRT2 when loading the PSL system. However, the subsequent execution of the PSL system (using the standard installation test data as input) was unsuccessful due to unidentified differences between the WWMCCS and GCOS environments.

One difference appears to exist in the dynamic program overlay operation in that when the usual SORT operation is omitted, correct operation of the CHANGE Function is obtained for a random-stored unit but incorrect operation is noted for an independent unit. The only significant divergence in program operations in these two cases is that the LKPCLU link is called in the former while the LKCHFC link is called in the latter. This noted divergence in CHLNE module operation appears to cause erroneous references to the File Description (FD) for the CHANGE-CARD-FILE (assigned to CH) such that "garbage" data is picked up and placed in the updated unit storage area. When the normal SORT operation is included (i.e., the LKSORT link is called in CHUNE module operation) erroneous data is obtained from the referenced CHANGE-CARD-FILE in all subsequent CHLNE module operations.

Examination of the PSL system load maps produced at the Andrews Air Force Base, WWMCCS facility and the RADC GCOS facility revealed no significant differences or sources of error. Since no difficulty is experienced in WWMCCS facility operations with this overlay procedure, it is assumed that an undetermined difference (or error) exists in GCOS facility operations.

Successful installation of the PSL system could not be completed at the RADC facility due to this undetermined difference.